深入理解 Magento – 第一章 – Magento 强大的配置系统

Magento 的配置系统就像是 Magento 的心脏，支撑着 Magento 的运行。这套配置系统掌管着几乎所有“module/model/class/template/etc”。它把整个 Magento 系统抽象出来，用一个配置文件来描述。这里的“配置文件”并不是一个物理上存在的文件，而是 Magento 根据当前的系统状态动态生成的一段 XML。大多数的 PHP 开发者并不习惯于这样抽象层，因为它增加的编程的复杂性。但是这样的抽象提供了无与伦比的灵活性，允许你覆盖几乎任何系统的默认行为。

首先，让我们写一个简单的插件来看看这个所谓的“配置文件”长什么样。虽然我已经提供的现成的代码，但是还是建议你自己建立这个插件，把整个流程走一遍有助于你的理解。

设置插件的目录结构

我们将要创建一个 Magento 的模块【注： Magento 的插件不叫 plug-in，叫 module，翻译成模块】。Magento 的模块由 php 和 xml 文件组成，目的是扩展或者覆盖系统的行为，比如为订单增加数据模型，更改一个类的方法，或者增加一个全新的功能。【注：Magento 自带的那些功能也都是基于模块的，比如用户注册，商品展示，结账流程等等。Magento 给我的感觉就是一切皆模块，和 Eclipse 的插件体系结构有点像】

大多数 Magento 的系统模块的结构和我们将要构建的插件的结构是一样的。Magento 的系统模块在以下目录

```
app/code/core/Mage
```

每一个子目录都是一个单独的模块。这些模块是由 Magento 官方开发的。我们安装完 Magento 以后，所使用的功能就是来自这些模块。我们自己创建的模块应该放在如下目录

```
app/code/local/Packagename
```

“Packagename”应该是一个唯一的字符串，用来标识你的代码。通常人们使用公司名字作为 Packagename，比如

```
app/code/local/Microsoft
```

由于我在做我自己的 Magento 项目，我将使用我自己的项目名“App”。 然后，我们要创建以下目录结构

```
app/code/local/App/Configviewer/Block
app/code/local/App/Configviewer/controllers
app/code/local/App/Configviewer/etc
app/code/local/App/Configviewer/Helper
app/code/local/App/Configviewer/Model
app/code/local/App/Configviewer/sql
```

你的插件并不一定需要包含以上所有的目录，但是为了以后开发方便，我们还是在一开始就把目录创建好。

接下来我们要创建两个文件，一个是 config.xml，放在 etc 目录下面

```
app/code/local/App/Configviewer/etc/config.xml
```

文件内容如下

```
<config>
    <modules>
        <App_Configviewer>
            <version>0.1.0</version>
```

```
        </App_Configviewer>
    </modules>
</config>
```

第二个文件需要在如下位置创建

```
app/etc/modules/App_Configviewer.xml
```

第二个文件应该遵循如下命名规则“Packagename_Modulename.xml”，文件内容如下

```
<config>
    <modules>
        <App_Configviewer>
            <active>true</active>
            <codePool>local</codePool>
        </App_Configviewer>
    </modules>
</config>
```

我们先不管这些文件是干什么的，以后会解释。建立好这两个文件以后，你的模块的骨架就已经完成了。Magento 已经知道你的模块存在，但是现在你的模块不会做任何事情。我们来确认一下 Magento 确实装载了你的模块

1. 清空 Magento 缓存
2. 在后台管理界面，进入 System->Configuration->Advanced
3. 展开“Disable Modules Output”
4. 确认“App_Configviewer”显示出来了

如果你看到“App_Configviewer”，那么恭喜你，你已经成功创建了你第一个 Magento 模块！

创建模块逻辑

我们之前创建的模块不会做任何事情，下面我们来为这个模块加入逻辑

1. 检查“showConfig”查询字符串是否存在
2. 如果“showConfig”存在，那么检查“showConfigFormat”查询字符串是否存在
3. 如果“showConfigFormat”存在，那么输出指定格式的配置信息，否则输出默认格式的配置信息
4. 终止执行流程

首先更改我们的 config.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <modules>
        <App_Configviewer>
            <version>0.1.0</version>
        </App_Configviewer>
    </modules>
    <global>
        <events>
            <controller_front_init_routers>
                <observers>
                    <app_configviewer_model_observer>
                        <type>singleton</type>
                        <class>App_Configviewer_Model_Observer</class>
```

```
                    <method>checkForConfigRequest</method>
                </app_configviewer_model_observer>
            </observers>
        </controller_front_init_routers>
    </events>
</global>
</config>
```

然后创建如下文件

```
App/Configviewer/Model/Observer.php
```

输入以下内容

```
<?php
class App_Configviewer_Model_Observer {
	const FLAG_SHOW_CONFIG = 'showConfig';
	const FLAG_SHOW_CONFIG_FORMAT = 'showConfigFormat';
	private $request;
	public function checkForConfigRequest($observer) {
		$this->request = $observer->getEvent ()->getData
( 'front' )->getRequest ();
		if ($this->request->{self::FLAG_SHOW_CONFIG} === 'true') {
			$this->setHeader ();
			$this->outputConfig ();
		}
	}
	private function setHeader() {
		$format = isset
( $this->request->{self::FLAG_SHOW_CONFIG_FORMAT} ) ?
$this->request->{self::FLAG_SHOW_CONFIG_FORMAT} : 'xml';
		switch ($format) {
			case 'text' :
				header ( "Content-Type: text/plain" );
				break;
			default :
				header ( "Content-Type: text/xml" );
		}
	}
	private function outputConfig() {
		die ( Mage::app ()->getConfig ()->getNode ()->asXML () );
	}
}
?>
```

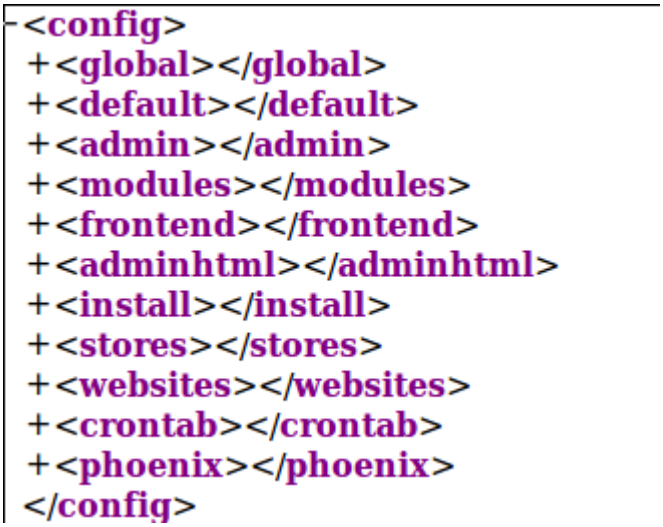
好了，代码编辑结束。清空你的 Magento 缓存，输入如下 URL

```
http://magento.example.com/?showConfig=true
```

【注： 根据文中的配置，不难看出任何指向 Magento 的 URL 加了“?showConfig=true”以后，都会输出同样的内容，正常的执行流程会被终止。】

配置文件分析

打开上述 URL，你应该看到一个巨大的 XML 文件。



这个文件描述了当前 Magento 系统的状态。它列出了所有的模块，数据模型，类，事件，监听器等等。举个例子，如果你搜索如下字符串

```
Configviewer_Model_Observer
```

你会发现刚刚你创建的那个类被列出来了。Magento 会解析每个模块的 config.xml，并把它们包含在这个全局配置中。

这个配置文件有啥用？

到目前为止，我们所作的事情似乎没什么意义，但是这个配置文件却是理解 Magento 的关键因素。你创建的每一个模块都会被加到这个配置文件中，任何时候，你需要调用一个系统功能的时候，Magento 都会通过这个配置文件来查询相应的模块和功能。举个简单的例子，如果你懂 MVC 的话，你应该和“helper class”之类概念的打过交道

```
$helper_salesrule = new Mage_SalesRule_Helper();
```

Magento 抽象了 PHP 的类声明方式。在 Magento 系统中，上面的代码等同于

```
$helper_salesrule = Mage::helper('salesrule');
```

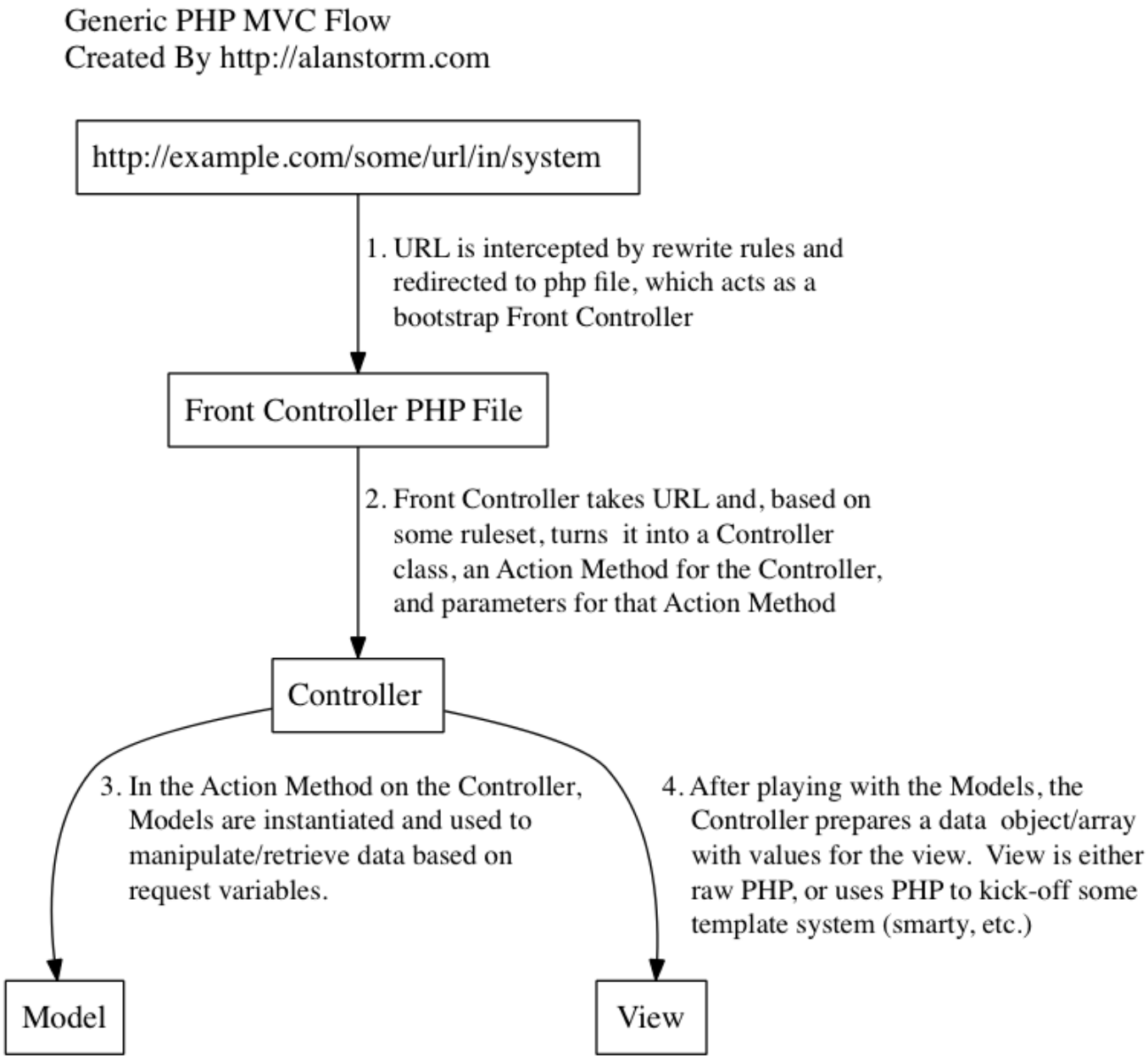
Magento 将通过以下逻辑来处理这行代码

1. 在配置文件中查找标签
2. 在里面查找 标签
3. 在里面查找 标签
4. 实例化从#3 找到的类（Mage_SalesRule_Helper）

Magento 总是通过配置文件来获得类名，这个逻辑看起来有些复杂，但这样做的优点也很明显，我们可以不需要更改 Magento 的代码就能更改 Magento 的核心功能。【注： 在这个例子中，我们可以通过修改配置文件用我们自己的 SalesRule_Helper 类来替换原来那个】这种高度抽象的编程方式在 php 中并不常见，但是它可以让你清晰的扩展或者替换系统的某一部分。

深入理解 Magento – 第二章 – Magento 请求分发与控制器

Model-View-Controller (MVC) ，模型-视图-控制器，源于 Smalltalk 编程语言和 Xerox Parc。现在有很多系统是基于 MVC 架构的，不同的系统 MVC 的实现也略有不同，但都体现了 MVC 的精髓，分离数据，业务逻辑和显示逻辑。最常见的 PHP MVC 框架是这样的



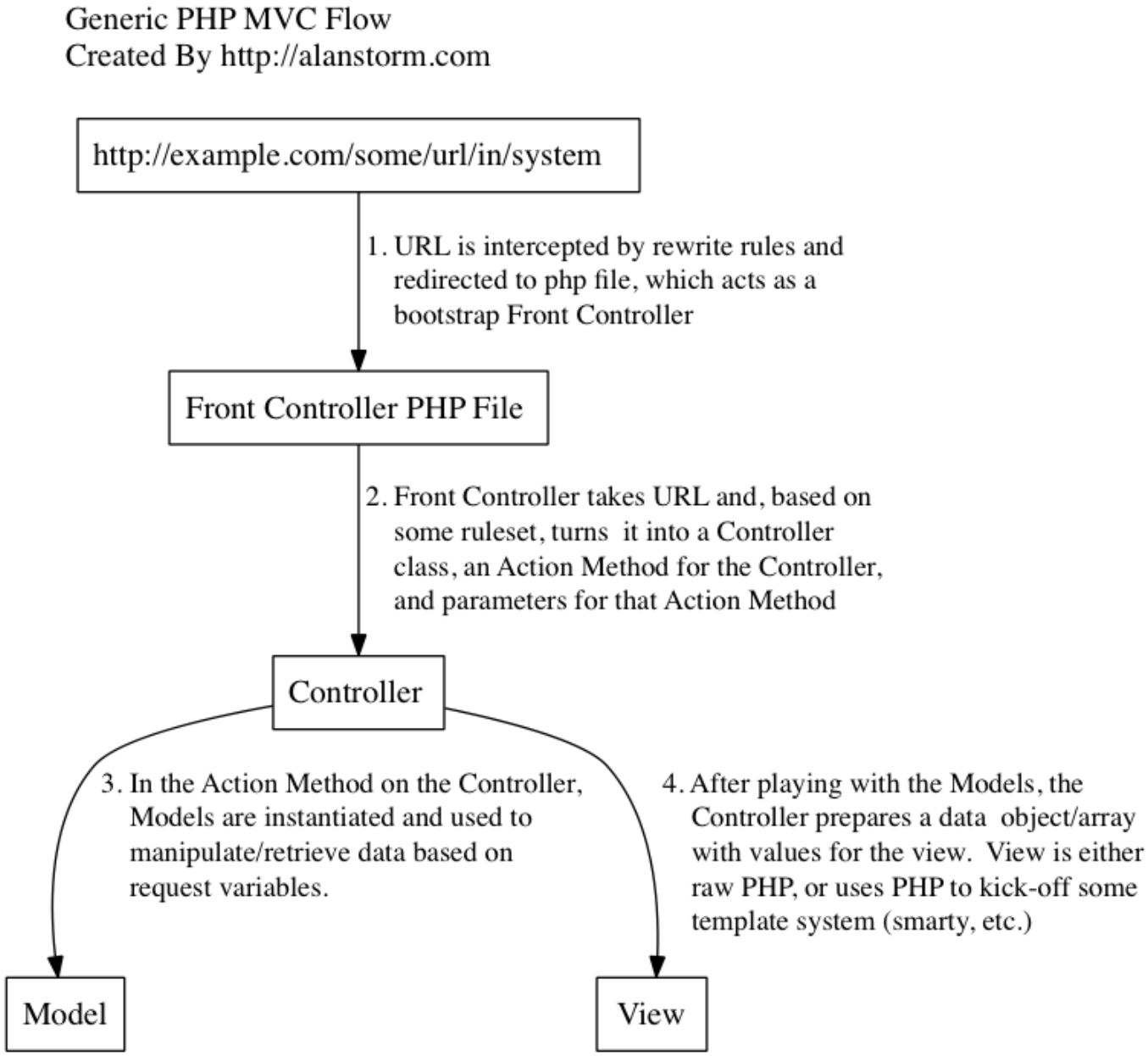
1. URL 请求被一个 PHP 文件拦截，通常称为前端控制器（Front Controller）
2. 这个 PHP 文件分析这个 URL，获得一个执行控制器（Action Controller）的名字和一个执行方法（Action Method）的名字，这个过程通常称为路由（Routing）
3. 实例化#2 获得的执行控制器
4. 调用执行控制器的执行方法
5. 执行方法中处理业务逻辑，比如获取数据
6. 执行控制器负责把数据传递给显示逻辑
7. 显示逻辑生成 HTML

这个架构相对于传统的“每个 php 都是一个页面”来讲已经是一个巨大的飞跃，但还是有人抱怨【注：CodeIgniter 就是这样一个 MVC 框架】

- 前端控制器仍然以全局的方式运行
- 基于配置的惯例导致了系统不够模块化

- URL Routing 不够灵活
- 控制器往往和视图绑定
- 更改默认设置往往导致大量的重构

Magento 创造了一个更抽象的 MVC 来解决上述问题。



1. URL 请求被一个 PHP 拦截
2. 这个 PHP 文件实例化一个 Magento 对象
3. Magento 对象实例化前端控制器
4. 前端控制器实例化全局配置中指定的路由对象，可以是多个
5. 路由对象会逐个与请求 URL 匹配
6. 如果发现匹配，那么可以获得一个执行控制器和一个执行方法的名字
7. 实例化#6 获得的执行控制器，并调用相应的执行方法
8. 执行方法中处理业务逻辑，模型数据
9. 控制器实例化布局对象（Layout）
10. 布局对象根据请求的参数，系统配置创建一个块对象（Block）列表，并实例化

11. 布局对象会调用块对象的 output 方法生成 HTML。这是一个递归的过程，因为块对象可以嵌套块对象
12. 每一个块对象都和一个模板文件（Template File）对应。块对象包含了显示逻辑，模板文件包含了 HTML 和 PHP 输出代码
13. 块对象直接从模型那里获得数据，换句话说，在 Magento 的 MVC 架构中，控制器并不直接把数据传给视图

这里很复杂，我们以后会详细解释每一个部分。我们先关注“前端控制器->路由对象->执行控制器”部分。

Hello World 示例

我们讲了太多理论，现在让我们来实践一下，通过实践来加深理解。下面是我们将要做的事情

1. 创建一个 Hello World 模块
2. 为这个模块配置路由
3. 为这个模块创建执行控制器

创建 Hello World 模块

首先，我们要创建一个模块的目录结构，这个我们以前已经做过了，就不再熬述

```
app/code/local/App/Helloworld/Block
app/code/local/App/Helloworld/controllers
app/code/local/App/Helloworld/etc
app/code/local/App/Helloworld/Helper
app/code/local/App/Helloworld/Model
app/code/local/App/Helloworld/sql
```

下面是 config.xml 的内容

```
PATH: app/code/local/App/Helloworld/etc/config.xml
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <modules>
        <App_Helloworld>
            <version>0.2.0</version>
        </App_Helloworld>
    </modules>
</config>
```

然后我们要创建一个系统配置文件来激活这个模块

```
PATH: app/etc/modules/App_Helloworld.xml
<config>
    <modules>
        <App_Helloworld>
            <active>true</active>
            <codePool>local</codePool>
        </App_Helloworld>
    </modules>
</config>
```

>

最后，让我们检查一下模块是不是已经被激活

1. 清空 Magento 缓存
2. 在管理后台，进入 System->Configuration->Advanced
3. 展开“Disable Modules Output”
4. 确认 App_Helloworld 显示出来了

配置路由

下面，我们要配置一个路由。路由是用来把一个 URL 请求转换成一个执行控制器和方法。和传统的 PHP MVC 不同的是，你需要在 Magento 的全局配置中显式的定义你的路由。我们继续上面的例子，在 config.xml 中，添加如下代码

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <modules>
        <App_Helloworld>
            <version>0.1.0</version>
        </App_Helloworld>
    </modules>
    <frontend>
        <routers>
            <helloworld>
                <use>standard</use>
                <args>
                    <module>App_Helloworld</module>
                    <frontName>helloworld</frontName>
                </args>
            </helloworld>
        </routers>
    </frontend>
</config>
```

在这里，我们有很多新名词要解释。

什么是 frontend？

frontend 标签指向一个 Magento 区（Area），比如“frontend”就是指网站的前台，“admin”是指网站的后台，“install”是指 Magento 的安装程序。【注：这个有点像磁盘分区，区和区之间是相互独立的，但是都归操作系统能够管理，在这里归 Magento 管理。默认的 Magento 安装没有“install”这个区，frontend 区接管了，全局配置中的以下代码可以解释这一点

```
<frontend>
        ...
          <install>
            <use>standard</use>
            <args>
              <module>Mage_Install</module>
              <frontname>install</frontname>
            </args>
          </install>
        ...
        </frontend>
```

】

什么是 routers？

Phil Karlton 有一句很著名的话“在计算机领域只有两件事是困难的：缓存和命名”。Magento 引入了很多新概念，无疑存在很多命名问题，这里就是一个例子。routers 标签有时候包含的是路由对象的定义，有时候包含的是路径的定义。路由对象是进行路由操作的实体，而路径仅仅是路由对象的一个参数。【注： 如果你仔细看过那个全局配置 xml 的话，你会发现有两处地方出现 routers，一处是“web -> routers”，另外一处是“frontend-> routers”。你再仔细看看会发现两处 routers 包含的内容不一样。第一处包含的是路由对象的定义，第二处包含的是路径的定义。】

什么是 module？

这个标签的内容应该是一个模块的全名，Packagename_Modulename，在这里是“App_Helloworld”。Magento 用这个名字来定位你的模块文件。

什么是 frontname？

当一个 router 解析一个 URL 的时候，它是按照如下规则进行的

```
http://example.com/frontName/actionControllerName/actionMethod/
```

所以，当我们在 frontname 标签里定义了“helloworld”以后，Magento 会把如下的 URL 请求交给我们的模块“App_Helloworld”来处理

```
http://example.com/helloworld/*
```

有些人容易把 frontname 和前端控制器（Front Controller）混淆起来。它们是两个不同的概念，frontname 只跟路由相关，学过 Zf 的人都知道，其实就是个模块名。【注： 根据我们前面讲过的 Magento 的 MVC 流程，前端控制器是用来实例化所有路由的，而这里的“frontName”只是路由过程中的一个参数】

什么是 helloworld？

这个标签的名字应该是模块名字的小写版本。我们的模块名字是“Helloworld”，所以这里我们用“helloworld”。你应该也已经注意到我们定义的“frontName”也是和我们的模块相匹配的。这是一个不成文的规定，但不是强制要求。事实上，一个模块可以定义多个，也就是可以有多个“frontName”。

为路由创建执行控制器

还记得 Magento 的 MVC 流程吗？路由会把控制权交给执行控制器。上面我们定义了路由，现在我们来定义我们的执行控制器。首先创建文件

```
app/code/local/App/Helloworld/controllers/IndexController.php
```

模块的控制器应该放在模块的子目录“controllers”（小写 c）里面。这是规定，Magento 会在这个目录寻找模块的控制器文件。我们的第一个控制器包含以下内容

```
class App_Helloworld_IndexController extends
Mage_Core_Controller_Front_Action {
    public function indexAction() {
        echo 'Hello World!';
    }
}
```

清空 Magento 缓存，请求如下 URL

```
http://exmaple.com/helloworld/index/index
```

如果你看到一个空白页面上面写着“Hello World”，那么恭喜你，你已经成功创建了你的第一个 Magento 控制器！

如何命名执行控制器？

还记得 config.xml 的标签吗？

```
<module>App_Helloworld</module>
```

执行控制的名字的构成如下

1. 以标签的内容开始 （App_Helloworld）
2. 紧接一个下划线 （App_Helloworld_）
3. 加上我们给控制器取的名字“Index”(App_Helloworld_Index)
4. 最后加上关键词“Controller” （App_Helloworld_IndexController）

我们自己定义的属于 frontend 区的执行控制器都应该继承 Mage_Core_Controller_Front_Action。

URL 里面的 index/index 是什么意思？

正如前文所述，Magento 默认的路由的规则如下

```
http://example.com/frontName/actionControllerName/actionMethod/
```

所以在我们请求的 URL

```
http://exmaple.com/helloworld/index/index
```

其中“helloworld”是“frontName”，第一个“index”是执行控制器（Action Controller）的名字，第二个“index”是执行方法的名字。对比我们写的执行控制器代码，我们不难发现执行方法的定义是执行方法名字加上“Action”关键字

```
public function indexAction(){...}
```

Magento 根据命名规则找到执行控制器文件并实例化，然后再根据命名规则调用指定的执行方法。如果 URL 没有给出执行控制器名字或者执行方法，Magento 会用默认的“index”来替代，所以下面三个 URL 是等价的

```
http://exmaple.com/helloworld/index/index
http://exmaple.com/helloworld/index/
http://exmaple.com/helloworld/
```

我们再来看一个例子。如果 URL 如下

```
http://exmaple.com/checkout/cart/add
```

Magento 的执行步骤如下

1. 查询全局配置，找到 frontName “checkout”对应的模块，Mage_Checkout
2. 找到执行控制器 “Mage_Checkout_CartController”
3. 调用执行控制器的“addAction”方法

进一步理解执行控制器

下面我们来为我们的执行控制器添加一个执行方法。添加如下代码到 IndexController.php

```
public function goodbyeAction() {
    echo 'Goodbye World!';
}
```

请求 URL

```
http://example.com/helloworld/index/goodbye
```

这次你应该看到“Goodbye World!”。因为我们继承了“Mage_Core_Controller_Front_Action”，我们可以使用一些父类已经定义好的方法和变量。比如父类会把 URL 后面跟的参数转换成 key/value 的数组。添加如下代码到我们的执行控制器

```
public function paramsAction() {
    echo '<dl>';
    foreach($this->getRequest()->getParams() as $key=>$value) {
        echo '<dt><strong>Param: </strong>'.$key.'</dt>';
        echo '</dl><dl><strong>Value: </strong>'.$value.'</dl>';
    }
    echo '';
}
```

请求如下 URL

```
http://example.com/helloworld/index/params?foo=bar&amp;baz=eof
```

你应该看到如下输出

Param: foo
Value: bar
Param: baz
Value: eof

最后，让我们再写一个执行控制器，用来处理以下 URL

```
http://example.com/helloworld/messages/goodbye
```

这里的执行控制器名字是“messages”，所以我们要创建如下文件

```
app/code/local/App/Helloworld/controllers/MessagesController.php
```

执行控制器的类名应该是

```
App_Helloworld_MessagesController
```

添加执行方法

```
public function goodbyeAction()
{
     echo 'Another Goodbye';
}
```

好了，Magento 的 MVC 架构大概就是这样了。它比传统的 PHP MVC 要复杂一点，但是 Magento 的这个高度灵活的 MVC 架构能让你创造出几乎所有你能想到的 URL 结构。

深入理解 Magento – 第三章 – Magento 的布局(Layout)，块(Block)和模板(Template)

我们接着研究 Magento。根据我们第二章讲的 Magento MVC 的架构，我们接下来应该讲模型（Model），但是我们跳过模型先来看布局和块。和一些流行的 PHP MVC 架构不同的是，Magento 的执行控制器不直接将数据传给试图，相反的视图将直接引用模型，从模型取数据。这样的设计就导致了视图被拆分成两部分，块（Block）和模板（Template）。块是 PHP 对象，而模板是原始 PHP 文件，混合了 XHTML 和 PHP 代码（也就是把 PHP 作为模板语言来使用了）。每一个块都和一个唯一的模板文件绑定。在模板文件 phtml 中，“$this”就是指该模板文件对应的快对象。

让我们来看一个例子

```
File:
app/design/frontend/base/default/template/catalog/product/list.phtml
```

你将看到如下代码

```
<?php $_productCollection=$this->getLoadedProductCollection() ?>
<?php if(!$_productCollection->count()): ?>
<p class="note-msg"><?php echo $this->__('There are no products matching
the selection.') ?></p>
<?php else: ?>
```

这里“getLoadedProductCollection”方法可以在这个模板的块对象“Mage_Catalog_Block_Product_List”中找到

```
File: app/code/core/Mage/Catalog/Block/Product/List.php
...
public function getLoadedProductCollection()
{
    return $this->_getProductCollection();
}
...
```

块的“_getProductCollection”方法会实例化模型，并读取数据然后返回给模板。

嵌套块

Magento 把视图分离成块和模板的真正强大之处在于“getChildHtml”方法。这个方法可以让你实现在块中嵌套块的功能。顶层的块调用第二层的块，然后是第三层……这就是 Magento 如何输出 HTML 的。让我们来看一下单列的顶层模板

```
File: app/design/frontend/base/default/template/page/1column.phtm
<?php
/**
 * Template for Mage_Page_Block_Html
 */
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php echo
$this->getLang() ?>" lang="<?php echo $this->getLang() ?>">
<head>
<?php echo $this->getChildHtml('head') ?>
</head>
<body<?php echo $this->getBodyClass()?'
class="'.$this->getBodyClass().'"':'' ?>>
<?php echo $this->getChildHtml('after_body_start') ?>
<div class="wrapper">
    <?php echo $this->getChildHtml('global_notices') ?>
    <div class="page">
        <?php echo $this->getChildHtml('header') ?>
        <div class="main-container col1-layout">
```

```
            <div class="main">
                <?php echo $this->getChildHtml('breadcrumbs') ?>
                <div class="col-main">
                    <?php echo $this->getChildHtml('global_messages') ?>
                    <?php echo $this->getChildHtml('content') ?>
                </div>
            </div>
        </div>
        <?php echo $this->getChildHtml('footer') ?>
        <?php echo $this->getChildHtml('before_body_end') ?>
    </div>
</div>
<?php echo $this->getAbsoluteFooter() ?>
</body>
</html>
```

我们可以看到这个模板里面很多地调用了“$this->getChildHtml(…)”。每次调用都会引入另外一个块的 HTML 内容，直到最底层的块。

布局对象

看到这里，你可能有这样的疑问

Magento 怎么知道在一个页面上要用那些块？

Magento 怎么知道哪一个块是顶层块？

“$this->getChildHtml(…)”里面的参数是什么意思？块的名字吗？

Magento 引入了布局对象（Layout Object）来解决上面的那些问题。布局对象（或者说布局文件）就是一个 XML 文件，定义了一个页面包含了哪些块，并且定义了哪个块是顶层块。

在第二章的时候我们在执行方法（Action Method）里面直接输出了 HTML 内容。现在我们要为我们的 Hello World 模块创建一个简单的 HTML 模板。首先我们要创建如下文件

```
app/design/frontend/default/default/layout/local.xml
```

包含以下内容

```
<?xml version="1.0" encoding="UTF-8"?>
<layout version="0.1.0">
    <helloworld_index_index>
        <reference name="root">
            <block type="page/html" name="root" output="toHtml"
template="helloworld/simple_page.phtml"/>
        </reference>
    </helloworld_index_index>
</layout>
```

再创建如下文件

```
app/design/frontend/default/default/template/helloworld/simple_page.
phtml
```

包含以下内容

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

```
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Untitled</title>
    <style type="text/css">
        body {
            background-color:#f00;
        }
    </style>
</head>
<body>
<h4>Links</h4>
<?php echo $this->getChildHtml('top.links'); ?>
<?php echo $this->getChildHtml('customer_form_register'); ?>
</body>
</html>
```

最后，我们要在执行控制器里面调用布局文件，开始输出 HTML。修改执行方法如下

```
public function indexAction() {
    //remove our previous echo
    //echo 'Hello Index!';
    $this->loadLayout();
    $this->renderLayout();
}
```

清空 Magento 缓存，访问 URL “http://exmaple.com/helloworld/index/index”。你应该看到一个纯红色背景的页面。这个页面的源代码应该和我们创建的文件“simple_page.phtml”一模一样。

究竟是怎么回事呢？

也许你看到这里一头雾水，没关系，我们来慢慢解释。首先你得安装一个 Layout Viewer 模块，这和我们第一章讲的 Config Viewer 模块很相似，都是查看 Magento 的内部信息。安装完这个模块之后【注：你需要参照第一章的内容，为这个模块创建“app/etc/modules/App_Layoutviewer.xml”】，打开如下 URL

```
http://example.com/helloworld/index/index?showLayout=page
```

你看到的是你正在请求的页面的布局文件。它是由 block，reference 和 remove 组成的。当你在执行方法中调用“loadLayout”时，Magento 会做如下处理

生成这个布局文件

为每一个 block 和 reference 标签实例化一个块对象。块对象的类名是通过标签的 name 属性来查找的。这些块对象被存储在布局对象的_blocks 数组中

如果 block 标签包含了 output 属性，那么这个块的名字和 output 属性的值会被添加到布局对象的_output 数组中

然后，当你在执行方法中调用“renderLayout”方法时，Magento 会遍历_output 数组中所有的块名字，从_blocks 数组中获得该名字的块，并调用块对象中使用 output 属性的值作为名字的函数。这个函数往往是“toHtml”。这个 output 属性也告诉 Magento 这里就是输出 HTML 的起点，也就是顶层块。【注：直接阅读 Layout 类的代码应该比较容易理解这里的逻辑

```
File: app/code/core/Mage/Core/Model/Layout.php
public function getOutput()
```

```
{
    $out = '';
    if (!empty($this->_output)) {
        foreach ($this->_output as $callback) {
            $out .= $this->getBlock($callback[0])->$callback[1]();
        }
    }
    return $out;
}
```

从这里我们也可以看出，一个页面的布局文件可以拥有多个顶层块。】

下面我们要讲解块对象是如何被实例化的，这个布局文件时如何被生成的，最后我们将动手做一个例子来实践这一章讲的内容。

实例化块对象

在布局文件中，block 和 reference 标签有一个“type”属性，这个属性其实是一个 URI

```
<block type="page/html" ...
<block type="page/template_links"...
```

Magento 就是通过这个 URI 是用来查找块对应的类名。这个 URI 分为两部分，第一部分“page”是用来在全局配置中查找一个基本类名，第二部分“html”或者“template_link”将被添加到基本类名后面生成一个具体的将被实例化的类名。

我们以“page/html”为例。首先 Magento 在全局配置中找到节点

```
/global/blocks/page
```

有以下内容

```
<page>
    <class>
        Mage_Page_Block
    </class>
</page>
```

这里我们拿到了一个基本类名“Mage_Page_Block”，然后添加 URI 的第二部分“html”到基本类名后面，我们就得到最终的块对象的类名“Mage_Page_Block_Html”。块的类名在 Magento 中被称为“分组类名”（Grouped Class Names），这些类都用相似的方法被实例化。我们将在以后的章节中详细介绍这个概念。

block 和 reference 的区别

我们上面提到 block 和 reference 都会实例化块对象，那么它们究竟有什么区别呢？ **reference 在布局文件中是用来表示替换一个已经存在的块**，举个例子

```
<block type="page/html" name="root" output="toHtml"
template="page/2columns-left.phtml">
<!-- ... sub blocks ... -->
</block>
<!-- ... -->
<reference name="root">
    <block type="page/someothertype" name="root"
template="path/to/some/other/template" />
    <!-- ... sub blocks ... -->
```

```
</reference>
```

Magento 首先创建了一个名叫“root”的块。然后，它有发现了一个引用（reference）的名字也叫“root”，Magento 会把原来那个“root”块替换成 reference 标签里面的那个快。

再来看看我们之前创建那个 local.xml

```
<layout version="0.1.0">
    <default>
        <reference name="root">
            <block type="page/html" name="root" output="toHtml"
template="helloworld/simple_page.phtml" />
        </reference>
    </default>
</layout>
```

在这里，块“root”被我们用 reference 替换了，指向了一个不同的模板文件。

布局文件是如何生成的

现在我们对布局文件已经有所了解了，但是这个布局文件是那里来的呢？要回答这个问题，我们得引入 Magento 中的另外两个概念，操作（Handle）和包布局（Package Layout）。

操作

Magento 会为每一个页面请求生成几个不同的操作。我们的 Layout View 模块可以显示这些处理器

```
http://example.com/helloworld/index/index?showLayout=handles
```

你应该看到类似如下列表的列表（和你的配置有关）

```
Handles For This Request
   1. default
   2. STORE_default
   3. THEME_frontend_default_gap
   4. helloworld_index_index
   5. customer_logged_out
```

它们每一个都是一个操作的名字。我们可以在 Magento 系统的不同的地方配置操作。在这里我们需要关注两个操作 “default” 和 “helloworld_index_index”。“default”处理器是 Magento 的默认处理器，参与每一个请求的处理。“helloworld_index_index”处理器的名字是 frontname “helloworld”加上执行控制器的名字“index”再加上执行方法的名字“index”。这说明执行控制器的每一个执行方法都有一个相应的操作。

我们说过“index”是 Magento 默认的执行控制器和执行方法的名字，所以以下请求的操作名字也是“helloworld_index_index”。

```
http://example.com/helloworld/?showLayout=handles
```

包布局

包布局和我们以前讲过的全局配置有些相似。它是一个巨大的 XML 文档包含了 Magento 所有的布局配置。我们可以通过以 Layout View 模块来查看包布局，请求一下 URL

```
http://example.com/helloworld/index/index?showLayout=package
```

你可能要等一会儿才能看到输出，因为文件很大。如果你的浏览器在渲染 XML 的时候卡死了，建议你换成 text 格式的

```
http://example.com/helloworld/index/index?showLayout=package&amp;showLayoutFormat=text
```

假设你选择的是 XML 格式输出，那么你应该看到一个巨大的 XML 文件，这就是包布局。这个文件是 Magento 动态生成的，合并当前主题（theme）下面所有的布局文件。如果你用的是默认安装的话，这些布局文件在以下目录

```
app/design/frontend/base/default/layout/
```

其实在全局配置中，有一个 updates 节点下面定义了所有将被装载的布局文件

```
<layout>
    <updates>
        <core>
            <file>core.xml</file>
        </core>
        <page>
            <file>page.xml</file>
        </page>
        ...
     </updates>
</layout>
```

当这些文件被装载以后，Magento 还会装载最后一个布局文件，helloworld.xml，也就是我们之前新建的那个文件。我们可以通过这个文件来定制 Magento 的布局。

结合操作和包布局

在包布局文件中，我们可以看到一些熟悉的标签 block，reference 等等，但是他们都包含在一下这些标签中

```
<default />
<catalogsearch_advanced_index />
etc...
```

这些就是操作标签。对于每个特定的请求来说，针对这个请求的布局文件是由包布局中所有和这个请求相关的操作标签组成的。比如我们上面的例子，和请求相关的操作标签如下

```
<default />
<store_bare_us />
<theme_frontend_default_default />
<helloworld_index_index />
<customer_logged_out />
```

所以，针对请求

```
http://example.com/helloworld/index/index
```

布局文件就是包布局中上面这些标签的内容组合。在包布局文件中，还有一个标签值得我们注意。我们可以通过这个标签引入另外一个操作标签。比如

```
<customer_account_index>
    <!-- ... -->
    <update handle="customer_account"/>
```

```
    !-- ... -->
</customer_account_index>
```

这段代码的意思是，如果一个请求包含了“customer_acount_index”操作，那么这个请求的布局文件也应该包含“customer_account”操作标签下面的 block 和 reference。

更新我们的例子

好了，理论讲完了，让我们来修改我们的例子，把这一章的内容实践一下。我们重新来看 local.xml

```
<layout version="0.1.0">
    <default>
        <reference name="root">
            <block type="page/html" name="root" output="toHtml"
template="helloworld/simple_page.phtml" />
        </reference>
    </default>
</layout>
```

我们用一个引用（reference）覆盖了名为“root”的块。然后定义了一个新的块，指向了一个不同的模板文件。我们把这个引用放在 default 操作标签下面，那就说明这个 Layout 将对所有的请求有效。如果你访问 Magento 自带的一些页面，你会发现它们要么是空白，要么就是和我们“hello world”例子的红色背景，但这并不是我们想要的效果。我们来修改一下 local.xml，让我们的模板仅对“helloworld”的请求有效。

```
<layout version="0.1.0">
    <helloworld_index_index>
        <reference name="root">
            <block type="page/html" name="root" output="toHtml"
template="helloworld/simple_page.phtml" />
        </reference>
    </helloworld_index_index>
</layout>
```

我们把操作标签换成了“helloworld_index_index”。清空 Magento 缓存，重新访问 Magento 的各个页面，你应该发现都恢复了正常，但是针对”hello world”模块的请求页面还是我们自定义的那个。

目前我们只实现了一个“index”执行函数，现在我们来实现“goodbye”执行函数。修改我们的执行控制器代码如下

```
public function goodbyeAction() {
    $this->loadLayout();
    $this->renderLayout();
}
```

但是你访问一下页面的时候你还是会看到 Magento 的默认布局

```
http://example.com/helloworld/index/goodbye
```

那是因为我们没有为这个请求定义布局。我们需要在 local.xml 中添加“helloworld_index_goodbye”标签。由于“index”请求和“goodbye”请求我们要套用的布局是一样的，所以我们将用标签来重用已有的配置

```
<layout version="0.1.0">
    <!-- ... -->
```

```
    <helloworld_index_goodbye>
        <update handle="helloworld_index_index" />
    </helloworld_index_goodbye>
</layout>
```

清空 Magento 缓存，请求以下 URL

```
http://example.com/helloworld/index/index
http://example.com/helloworld/index/goodbye
```

你将会得到两个完全相同的页面。

开始输出和 getChildHtml 方法

在 Magento 默认的配置下，HTML 输出是从名为“root”的块开始（其实是因为这个块拥有 output 属性【注：任何一个拥有 output 属性的块都是顶层块，在拥有多个顶层块的情况下 Magento 将按照块定义的先后顺序输出 HTML】）。我们覆盖了“root”块的模板

```
template="helloworld/simple_page.phtml"
```

模板文件的查找路径是当前主题（theme）的根目录，Magento 默认设置时这里

```
app/design/frontend/base/default
```

为页面加入内容

到目前为止，我们的页面都比较无聊，啥也没有。我们来为页面加点有意义的内容。修改 local.xml 如下

```
<helloworld_index_index>
    <reference name="root">
        <block type="page/html" name="root"
template="helloworld/simple_page.phtml">
            <block type="customer/form_register"
name="customer_form_register"
template="customer/form/register.phtml"/>
        </block>
    </reference>
</helloworld_index_index>
```

我们在“root”块里面嵌套了一个块“customer_form_register”。这个块是 Magento 本来就有的，包含了一张用户注册表单。我们把这个块嵌套进来，那么我们在模板文件里面就能用这个块的内容。使用方法如下，修改 simple_page.phtml

```
<body>
    <?php echo $this->getChildHtml('customer_form_register'); ?>
</body>
```

这里“getChildHtml”的参数就是要引入的块的名字，使用起来相当方便。清空 Magento 缓存，刷新 hello world 页面，你应该在红色背景上看到用户注册表单。Magento 还有一个块，叫做“top.links”，让我们把它也加进来。修改 simple_page.html

```
        <body>
          <h1>Links</h1>
          < ?php echo $this->getChildHtml('top.links'); ?>
```

```
            < ?php echo $this->getChildHtml('customer_form_register'); ?>
        </body>
```

刷新页面，你会发现 Links 显示出来了，但是“top.links”什么都没有显示。那是因为我们并没有把这个块引入到 local.xml，所以 Magento 找不到这个块。“getChildHtml”的参数一定要是当前页面的布局文件中声明过的块。这样的话 Magento 就可以只实例化需要用到的块，节省了资源，我们也可以根据需要为块设置不同的模板文件。

我们修改 helloworld.xml 文件如下

```
<?xml version="1.0" encoding="UTF-8"?>
<layout version="0.1.0">
    <helloworld_index_index>
        <reference name="root">
            <block type="page/html" name="root" output="toHtml"
template="helloworld/simple_page.phtml">
                <block type="customer/form_register"
name="customer_form_register"
template="customer/form/register.phtml"/>
                <block type="page/template_links" name="top.links"/>
            </block>
        </reference>
    </helloworld_index_index>
</layout>
```

清空 Magento 缓存，刷新页面，你会看到一排链接显示出来了。【注：如果你细心一点的话你会发现“top.links”块没有 template 属性，那是因为这个块的类中一定定义了默认的模板

```
protected function _construct()
{
    $this->setTemplate('page/template/links.phtml');
}
```

】

总结

这一章我们讲解了布局的基础知识。你可能会觉得这个很复杂，但是你也不必过分担心，因为平常使用 Magento 是不会用到这些知识的，Magento 提供的默认布局应该可以满足大部分需求。对于想要深入研究 Magento 的开发者来说，理解 Magento 的布局是至关重要的。布局，块和模板构成了 Magento MVC 架构中的 View，这也是 Magento 的特色之一。

深入理解 Magento – 第四章 – 模型和 ORM 基础

对于任何一个 MVC 架构，模型（Model）层的实现都是占据了很大一部分。对于 Magento 来说，模型占据了一个更加重要的位置，因为它常常包含了一部分商业逻辑代码（可以说它对，也可以说它错）。这些代码在其他的 MVC 框架中往往出现在控制器或者帮助函数中。

传统的 PHP MVC 架构中的模型

本来 MVC 的定义就不是很清晰，不同的人有不同的看法，而对于模型的定义争议就更多了。在 MVC 模式被广泛采用之前，PHP 程序员往往通过 SQL 语句直接操作数据库。也有些程序员通过一个 SQL 抽象层来操作数据库（比如 AdoDB）。程序员往往关注 SQL 语句本身，而不是和数据相关的对象。

虽然直接操作 SQL 的方式一直被病诟，但是很多 PHP 框架还是以 SQL 为中心的。模型层提供了一系列对象，抽象/封装了数据操作，但是程序员最终还是需为模型层对象写 SQL 语句操作数据库。
还有一些框架回避了 SQL，使用了对象关系映射（Object Relational Mapping，ORM）来解决这个问题。使用这个方法的话，程序员不用关注 SQL，而只需要和对象打交道。我们可以操作一个对象的属性，当“Save”方法被调用的时候，对象的属性会作为数据自动的被写入数据库。有些 ORM 框架会根据数据表的信息自动推测对象的属性，也有框架要求用户显示的生命对象属性和表的关系。比较有名的 ORM 框架有 ActiveRecord 等等。【注：ActiveRecord 源自 Ruby on Rails，不过现在 PHP 也有了】
关于 ORM 的概念，我就解释到这里。但是和许多计算机领域的其他概念一样，ORM 的定义也越来越模糊了。我不想在这片文章中讨论关于 ORM 的争议，所以我说的 ORM 就是那个最基本的 ORM 概念。

Magento 的模型

Magento 理所当然的也追随潮流应用了 ORM。虽然 Magento 自带的 Zend 框架提供了 SQL 抽象层，但是在大多数情况下我们将通过 Magento 自带的模型和我们自己的模型来进行数据访问。他和视图层（View）一样，Magento 的模型层也不是简单的 ORM，而是一个高度灵活，高度抽象甚至有点令人费解。

解剖 Magento 的模型

大部分的 Magento 模型分为两类。第一类是基本的 ActiveRecord 类型，一张表一个对象的模型。第二类是 Entity Attribute Value（EAV）模型。【注：EAV 翻译成“实体属性值”有点词不达意，还是就叫 EAV 的好】
Magento 自己定义了一个数据类型叫做模型集合（Model Collection）。顾名思义，模型集合就是一个对象里面包含了很多模型对象。Magento 的创造者 Varien 团队实现了 PHP 类库的标准接口，“IteratorAggregate”，“Countable”。这样模型集合就能调用这些方法，这也是模型集合和数组的区别。
Magento 的模型并不直接访问数据库。每一个模型都有一个资源模型（Resource Model），每一个资源模型拥有两个适配器（Adapter），一个读，一个写。这样的话逻辑模型和数据库访问就分开了，所以从理论上讲更改底层数据库只需要重写适配器就可以了，所有上层代码都不需要更改。
创建一个基本模型
继续我们 Hello World 的例子。在 Hello World 模块中创建 BlogController.php 如下

```
class App_Helloworld_BlogController extends
Mage_Core_Controller_Front_Action {
    public function indexAction()
    {
        echo 'Hello Blog';
    }
}
```

访问以下 URL

```
http://127.0.0.1/Magento/helloworld/blog
```

你应该看到“Hello Blog”输出。

创建数据表

我们可以通过 Magento 自带的方法创建或者修改数据库，但是为了不引入过多新内容，我们暂且手工创建一张表。在你的数据库中执行以下语句

```
CREATE TABLE `blog_posts` (
    `blogpost_id` int(11) NOT NULL AUTO_INCREMENT,
    `title` text,
    `post` text,
    `date` datetime DEFAULT NULL,
    `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (`blogpost_id`)
```

```
);
INSERT INTO `blog_posts` VALUES (1,'My New Title','This is a blog
post','2009-07-01 00:00:00','2009-07-02 23:12:30');
```

这里我们创建了一张名为“blog_posts”的表，并填充了一条数据。

创建模型

要设置一个模型一共有以下四个步骤

1. 启用模型
2. 启用资源模型
3. 在资源模型中添加实体（Entity）。对于简单的模型来说，实体就是数据表的名字
4. 为资源模型设置读、写适配器

在进行这些步骤之前，我们先来看假设这些步骤已经做完了，我们怎么用一个模型。在 Magento 中，我们用以下的方式来实例化一个模型

```
$model = Mage::getModel('helloworld/blogpost');
```

和我们以前讲过的“Mage::getHelper()”的原理类似，这里 Magento 也是通过全局配置去查找模型的类名。模型的类名和我们以前讲过的块类名一样，都是分组类名。这里参数的前半部分“helloworld”是组名（Group Name），后半部分“blogpost”是半类名（Class Name）【注：我将“Class Name”翻译成半类名是为了和类名区分开来】。具体步骤如下

1. 从全局配置“/global/models/GROUP_NAME/class”获得基本类名“App_Helloworld_Model”
2. 检查全局配置“/global/models/GROUP_NAME/rewrite/CLASS_NAME”是否设置，如果有那么这个节点的值将被作为类名实例化
3. 否则，最终的类名将是基本类名加上半类名，也就是“App_Helloworld_Model_Blogpost”

启用模型

修改模块的 config.xml

```
<global>
<!-- ... -->
<models>
    <helloworld>
        <class>App_Helloworld_Model</class>
        <!--
            need to create our own resource, can't just
            use core_mysql4
            -->
        <resourceModel>helloworld_mysql4</resourceModel>
    </helloworld>
    </models>
    <!-- ... -->
</global>
```

标签就是组名，也应该和模块名一致。标签的内容是基本类名，所有 Helloworld 模块的模型都用这个基本类名,命名方式如下

```
Packagename_Modulename_Model
```

标签指明了这个模块的模型要用哪个资源模型。这个标签的内容是组名加上“mysql4”我们将在后面详细介绍资源模型。

现在让我们来实例化一个模型看看，修改 indexAction 方法

```
public function indexAction() {
    $blogpost = Mage::getModel('helloworld/blogpost');
    echo get_class($blogpost);
}
```

清空 Magento 缓存，刷新页面，你应该看到一个类似这样的异常（请先打开 Magento 的开发模式）

```
include(App\Helloworld\Model\Blogpost.php) [<a
href='function.include'>function.include</a>]: failed to open stream:
No such file or directory
```

原因很简单，就是 Magento 尝试去实例化“App_Helloworld_Model_Blogpost”，但是它在 Helloworld 模块的文件夹里面找不到这个类。所以我们现在来创建这个类

```
      File: app/code/local/App/Helloworld/Model/Blogpost.php
      class App_Helloworld_Model_Blogpost extends
Mage_Core_Model_Abstract
      {
          protected function _construct()
          {
              $this->_init('helloworld/blogpost');
          }
      }
```

刷新页面，你应该看到页面上显示”App_Helloworld_Model_Blogpost”。所有的模型都必须继承“Mage_Core_Model_Abstract”类。这个抽象类强制你实现一个方法“_construct”（注意：这个不是 PHP 的构造行数“__construct”）。这个方法应该调用父类已经定义好的“_init”方法，参数是资源模型的 URI，也就是我们要告诉模型使用哪个资源模型。我们将在解释资源模型的时候再解释这个 URI。

启用资源模型并添加实体

好了，我们设置好了模型，下面我们要为模型设置资源模型。资源模型才是真正和数据库对话的组件。在模型的配置中，有一段这样的代码

```
<resourcemodel>helloworld_mysql4</resourcemodel>
```

的值将被用来实例化资源模型。我们不需要显式的调用资源模型，但是当一个模型需要访问数据库的时候，Magento 会自动实例化一个资源模型来使用。

```
Mage::getResourceModel('helloworld/blogpost');
```

这里“helloworld/blogpost”就是我们给模型的“_init”传入的参数。“helloworld”是组名，“blogpost”是模型的半类名。“Mage::getResourceModel”方法将以“helloworld/blogpost”为 URI 在全局配置中找到标签的值，在这里是“helloworld_mysql4”。然后 Magento 会用 URI“helloworld_mysql4/blogpost”去实例化资源模型类。实例化的过程和我们前面讲的模型的实例化是一样的，所以我们也需要在 config.xml 中添加资源模型的声明

```
<global>
<!-- ... -->
<models>
<!-- ... -->
<helloworld_mysql4>
```

```
    <class>App_Helloworld_Model_Resource_Mysql4</class>
</helloworld_mysql4>
</models>
</global>
```

这里我们可以看到，资源模型的声明也是放在下面的。有点搞，但是也不必深究了，Magento 就这么定义的。标签的值是所有资源模型类的基本类名，命名方式如下

```
Packagename_Modulename_Model_Resource_Mysql4
```

好了，我们已经配置了资源模型，我们来试试装载一些数据。修改 indexAction 如下

```
        public function indexAction() {
            $params = $this->getRequest()->getParams();
            $blogpost = Mage::getModel('helloworld/blogpost');
            echo("Loading the blogpost with an ID of ".$params['id']."<br
/>");
            $blogpost->load($params['id']);
            $data = $blogpost->getData();
            var_dump($data);
        }
```

清空 Magento 缓存，访问下面的页面

```
http://127.0.0.1/Magento/helloworld/blog/index/id/1
```

你应该看到一个类似下面这样的异常

```
include(App\Helloworld\Model\Resource\Mysql4\Blogpost.php)
[function.include]: failed to open stream: No such file or directory
```

我想你看到这里也明白了，我们要为模型添加一个资源类，添加如下文件

```
File:
app/code/local/App/Helloworld/Model/Resource/Mysql4/Blogpost.php
      class App_Helloworld_Model_Resource_Mysql4_Blogpost extends
Mage_Core_Model_Mysql4_Abstract{
          protected function _construct()
          {
              $this->_init('helloworld/blogpost', 'blogpost_id');
          }
      }
```

这里“_init”方法的第一个参数这个资源模型将要使用的数据表的 URI，第二个参数是数据表中的列名。这个列的内容必须唯一，往往是数据表的主键。

为资源模型添加实体

刷新页面，你是不是得到下面的异常？

```
Can't retrieve entity config: helloworld/blogpost
```

那是因为我们的资源文件现在还是一个空壳，并没有和数据库联系起来。现在我们来把资源模型和我们的表联系起来，修改 config.xml 如下

```
<global>
      <!-- ... -->
      <models>
         <!-- ... -->
         <helloworld_mysql4>
            <class>App_Helloworld_Model_Resource_Mysql4</class>
            <entities>
               <blogpost>
                  <table>blog_posts</table>
               </blogpost>
            </entities>
         </helloworld_mysql4>
      </models>
</global>
```

我们前面设置了资源模型使用的数据表的 URI 是“helloworld/blogpost”，那么 Magento 会把“helloworld”作为组名，“blogpost”作为实体名，也就是。在 Magento 的简单模型中（也就是继承 Mage_Core_Model_Mysql4_Abstract 的模型），一个实体对应一张数据表。我们的数据表是“blog_posts”，所以这里
标签的内容就是“blog_posts”。
清空 Magento 缓存，再次刷新页面，你应该看到以下内容

```
Loading the blogpost with an ID of 1
      array(5) { ["blogpost_id"]=> string(1) "1" ["title"]=> string(12)
"My New Title" ["post"]=> string(19) "This is a blog post" ["date"]=>
string(19) "2009-07-01 00:00:00" ["timestamp"]=> string(19) "2009-07-02
23:12:30" }
```

设置读写适配器

在上面的例子中，我们已经可以从数据库中取数据了，但是我们却没有为资源模型设置读写适配器，怎么回事呢？原因很简单，那就是因为 Magento 会为没有适配器的资源模型启用默认适配器。我们也可以显式的配置默认的适配器

```
<global>
      <!-- ... -->
      <resources>
         <helloworld_write>
            <connection>
               <use>default_write</use>
            </connection>
         </helloworld_write>
         <helloworld_read>
            <connection>
               <use>default_read</use>
            </connection>
         </helloworld_read>
      </resources>
```

```
</global>
```

在标签下面有两个部分，一个读，一个写。标签名字中的“hellworld”是我们定义的组名【注：在资源模型的“_init”函数中传入的数据表的 URI “helloworld/blogpost”的前半部分就是适配器名字的前半部分】。从这里我们也可以看出来一个资源组对应一对适配器。清空 Magento 缓存，刷新浏览器，你应该看到和刚才相同的页面。【注：如果你去全局配置中找“core_read”你会发现“default_read”，然后是“default_setup”

```
<default_setup>
      <connection>
         <model>mysql4</model>
         <initstatements>SET NAMES utf8</initstatements>
         <type>pdo_mysql</type>
         <host>localhost</host>
         <username>root</username>
         <password>admin</password>
         <dbname>zend-magento</dbname>
         <active>1</active>
      </connection>
</default_setup>
```

这才是最终和数据库连接的详细信息。如果你再往下深究，你会发现全局配置有这么一段

```
<resource>
      <connection>
         <types>
            <pdo_mysql>

<class>Mage_Core_Model_Resource_Type_Db_Pdo_Mysql</class>
            </pdo_mysql>
         </types>
      </connection>
</resource>
```

所以，“Mage_Core_Model_Resource_Type_Db_Pdo_Mysql”才是最终连接数据库的类。如果我们更换数据库的话，我们要重写一个相似的类来连接别的数据库。

】

基本模型操作

所有的模型最终都继承自类“Varien_Object”。这个类属于 Magento 的系统类库，不属于 Magento 的核心模块。你可以在以下位置找到这个类

```
lib/Varien/Object.php
```

Magento 模型的数据保存在“_data”属性中，这个属性是“protected”修饰的。父类“Varian_Object”定义了一些函数用来取出这些数据。我们上面的例子用了“getData”，这个方法返回一个数组，数组的元素是“key/value”对。【注：其实就是数据表中一行的数据，“key”就是列名，“value”就是值】我们可以传入一个参数获取某个具体的“key”的值。

```
$model->getData();
      $model->getData('title');
```

还有一个方法是“getOrigData”，这个方法会返回模型第一次被赋予的值。【注：因为模型在初始化以后，值可以被修改，这个方法就是拿到那个最原始的值】

```
$model->getOrigData();
       $model->getOrigData('title');
```

“Varien_Object”也实现了一些 PHP 的特殊函数，比如神奇的“__call”。你可以对任何一个属性调用“get, set, unset, has”方法

```
$model->getBlogpostId();
       $model->setBlogpostId(25);
       $model->unsetBlogpostId();
       if($model->hasBlogpostId()){...}
```

这里的方法名中的属性名字符合“camelcase”命名规则【注：简单的说就是 Java 的命名规则，每个单词的第一个字母大写，第一个字母可以大写也可以小写】。为了有效的利用这些方便的方法，我们在定义数据表列名的时候要用小写，并用下划线作为分隔符，比如“blogpost_id”。在最近的 Magento 版本中，这个规则已经被弱化，为了实现 PHP 的“ArrayAccess”接口

```
$id = $model->['blogpost_id'];
       $model->['blogpost_id'] = 25;
        //etc...
```

也就是说，你会在 Magento 中同时看到这两种技巧的使用。

Magento 中的 CRUD 操作

Magento 模型通过“load, save, delete”三个方法来支持基本的 Create，Read，Update 和 Delete 操作。我们在上面已经使用过“load”方法了。这个方法的参数就是要装在的数据记录的“id”。

```
$blogpost->load(1);
```

“save”方法可以用来创建新数据或者修改已有数据。我们在 BlogController.php 中添加如下方法

```
public function createNewPostAction() {
          $blogpost = Mage::getModel('helloworld/blogpost');
          $blogpost->setTitle('Code Post!');
          $blogpost->setPost('This post was created from code!');
          $blogpost->save();
          echo 'post created';
       }
```

访问以下 URL

```
http://127.0.0.1/Magento/helloworld/blog/createNewPost
```

现在你数据表中应该有两条数据了。下面来修改一条数据

```
       public function editFirstPostAction() {
          $blogpost = Mage::getModel('helloworld/blogpost');
          $blogpost->load(1);
          $blogpost->setTitle("The First post!");
          $blogpost->save();
          echo 'post edited';
```

```
    }
```

最后，我们来删除一条数据

```
    public function deleteFirstPostAction() {
        $blogpost = Mage::getModel('helloworld/blogpost');
        $blogpost->load(1);
        $blogpost->delete();
        echo 'post removed';
    }
```

模型集合

上面的例子我们只是演示了对单个数据操作，现在我们来看看如何同时操作多条记录。我们上面已经讲过，每个 Magento 的模型都有一个独特的模型集合。这些模型集合实现了 PHP 的“IteratorAggregate”和“Countable”接口，也就是他们可以作为“count”函数的参数，并且可以在“for each”语句中使用。
现在让我们来看看如何使用模型集合，在 Blog 控制器中添加如下方法

```
    public function showAllBlogPostsAction() {
        $posts =
Mage::getModel('helloworld/blogpost')->getCollection();
        foreach($posts as $blog_post){
            echo '<h3>'.$blog_post->getTitle().'</h3>';
            echo nl2br($blog_post->getPost());
        }
    }
```

访问如下 URL

```
http://127.0.0.1/Magento/helloworld/blog/showAllBlogPosts
```

你应该看到以下异常

```
include(App\Helloworld\Model\Resource\Mysql4\Blogpost\Collection.php)
[function.include]: failed to open stream: No such file or directory
```

我想你不会被这个异常吓到，已经熟门熟路了。我们需要添加一个 PHP 类，定义 Blogpost 的模型集合。每个模型都有一个“protected”属性“_resourceCollectionName”【注：从父类“Mage_Core_Model_Abstract”继承来的】。这个属性的值是这个模型对应的模型集合的 URI。

```
protected '_resourceCollectionName' => string
'helloworld/blogpost_collection'
```

在默认情况下，这个值是模型的 URI 加上“_collection”。Magento 把模型集合也看做是一种资源（Resrouce），所以运用资源模型的命名规则，模型集合的全名是

```
App_Helloworld_Model_Resource_Mysql4_Blogpost_Collection
```

然后我们要创建如下文件

```
File:
app/code/local/App/Helloworld/Model/Resource/Mysql4/Blogpost/Collectio
n.php
class App_Helloworld_Model_Resource_Mysql4_Blogpost_Collection extends
```

```
Mage_Core_Model_Mysql4_Collection_Abstract {
    protected function _construct()
    {
            $this->_init('helloworld/blogpost');
    }
}
```

这里的参数是模型的 UR，用来 I 来初始化模型集合。刷新页面，你应该看到数据库中的 Blog 都显示出来了。

总结

首先我要恭喜你，到这里你已经创建并配置了你的第一个 Magento 模型。在后面的章节中我们将讲解 Magento 的另外一种模型 Entity Attribute Value Model。

在这章开始的时候，我撒了一个小谎。其实在 Magento 中，并不是所有的模型都继承自“Mage_Core_Model_Abstract”。在 Magento 最初的版本中，这个抽象类并不存在。所以有很多模型是直接继承自“Varien_Object”。不过这些并不影响我们创建 Magento 模型，了解一下就可以了，方便阅读 Magento 的代码。

深入理解 Magento – 第五章 – Magento 资源配置

对于任何一个更新频繁的项目来说，保持开发环境和生产环境的数据库同步是件很头疼的事情。Magento 提供了一套系统，用版本化的资源迁移脚本来解决这个问题。

上一章，我们为 Helloworld Blogpost 创建了一个模型。我们直接通过 SQL 语句“CREATE TABLE”来创建数据表。在这一章，我们将为 Helloworld 模块创建一个资源配置（Setup Resource）用于创建数据表。我们也会创建一个模块升级脚本，用来升级已经安装的模块。下面是我们要做的步骤

1. 在配置文件中添加资源配置
2. 创建资源类文件
3. 创建安装脚本
4. 创建升级脚本
5. 添加资源配置

修改 Helloworld 模型的 config.xml

```
<resources>
    <!-- ... -->
    <helloworld_setup>
        <setup>
            <module>Zhlmmc_Helloworld</module>
            <class>Zhlmmc_Helloworld_Model_Setup_Mysql4_Setup</class>
        </setup>
        <connection>
            <use>core_setup</use>
        </connection>
    </helloworld_setup>
    <!-- ... -->
</resources>
```

标签是用来唯一标识我们正在创建的资源配置。虽然不是强制要求，但是我们应该使用 “modelname_setup” 来命名资源配置。标签的内容是“Packagename_Modulename”。最后，标签的内容就是我们将要创建的资源配置类的类名。虽然对于基本的配置来说，没有必要创建一个单独的资源配置类，但是为了更好的理解资源

配置是如何工作的，我们的例子还是创建一个单独的类。

File: app/code/local/Zhlmmc/Helloworld/Model/Setup/Mysql4/Setup.php

```
class Zhlmmc_Helloworld_Model_Setup_Mysql4_Setup extends Mage_Core_Model_Resource_Setup {
}
```

创建安装脚本

下面我们将要创建一个安装脚本。这个安装脚本包含了“CREATE TABLE”等 SQL 语句。这个脚本将在模块初始化的被运行。首先我们来看一下模块的配置文件

```
<modules>
    <zhlmmc_helloworld>
        <version>0.1.0</version>
    </zhlmmc_helloworld>
</modules>
```

这一部分是所有 config.xml 都必须包含的。它包含了模块的名称，还有版本。我们的安装脚本的名字将基于这个版本号，“0.1.0”。创建以下文件

File: app/code/local/Zhlmmc/Helloworld/sql/helloworld_setup/mysql4-install-0.1.0.php

```
echo 'Running This Upgrade: '.get_class($this)."\n   \n";
die("Exit for now");
```

文件路径中的“helloworld_setup”应该和上文在 config.xml 中添加的一致。文件名中的“0.1.0”就是模块的版本号。清空 Magento 缓存，访问任何 URL，你应该看到以下内容

Running This Upgrade: Zhlmmc_Helloworld_Model_Setup_Mysql4_Setup

Exit for now

…

这说明我们的安装脚本已经被运行了。我们先不放 SQL 脚本在这里，先把创建一个资源配置的流程走完。移除“die()”语句，重新装载页面，你应该看到你的 Upgrade 语句在页面的顶部，再次刷新页面，页面应该正常显示了。

资源版本

Magento 的资源配置系统允许你直接拷贝安装脚本和升级脚本到服务器上，Magento 会根据当前模块的版本自动运行相应的脚本。这样你就只需要维护一份数据库迁移脚本。我们先来看看“core_resource”数据表

```
mysql> select code,version from core_resource;
+----------------------+------------+
| code                 | version    |
+----------------------+------------+
| adminnotification_setup | 1.0.0      |
| admin_setup          | 0.7.2      |
| alipay_setup         | 0.9.0      |
| api_setup            | 0.8.1      |
| backup_setup         | 0.7.0      |
| bundle_setup         | 0.1.11     |
| canonicalurl_setup   | 0.1.0      |
| catalogindex_setup   | 0.7.10     |
| cataloginventory_setup | 0.7.5      |
| catalogrule_setup    | 0.7.8      |
| catalogsearch_setup  | 0.7.7      |
| catalog_setup        | 1.4.0.0.21 |
```

```
| checkout_setup         | 0.9.5     |
| chronopay_setup        | 0.1.0     |
| cms_setup              | 0.7.13    |
| compiler_setup         | 0.1.0     |
| contacts_setup         | 0.8.0     |
| core_setup             | 0.8.26    |
| cron_setup             | 0.7.1     |
| customer_setup         | 1.4.0.0.6 |
| cybermut_setup         | 0.1.0     |
| cybersource_setup      | 0.7.0     |
| dataflow_setup         | 0.7.4     |
| directory_setup        | 0.8.10    |
| downloadable_setup     | 0.1.16    |
| eav_setup              | 0.7.15    |
| eway_setup             | 0.1.0     |
| flo2cash_setup         | 0.1.1     |
| giftmessage_setup      | 0.7.2     |
| googleanalytics_setup  | 0.1.0     |
| googlebase_setup       | 0.1.1     |
| googlecheckout_setup   | 0.7.3     |
| googleoptimizer_setup  | 0.1.2     |
| helloworld_setup       | 0.1.0     |
| ideal_setup            | 0.1.0     |
| index_setup            | 1.4.0.2   |
| log_setup              | 0.7.7     |
| moneybookers_setup     | 1.2       |
| newsletter_setup       | 0.8.2     |
| oscommerce_setup       | 0.8.10    |
| paybox_setup           | 0.1.3     |
| paygate_setup          | 0.7.1     |
| payment_setup          | 0.7.0     |
| paypaluk_setup         | 0.7.0     |
| paypal_setup           | 0.7.4     |
| poll_setup             | 0.7.2     |
| productalert_setup     | 0.7.2     |
| protx_setup            | 0.1.0     |
| rating_setup           | 0.7.2     |
| reports_setup          | 0.7.10    |
| review_setup           | 0.7.6     |
| salesrule_setup        | 0.7.12    |
| sales_setup            | 0.9.56    |
| sendfriend_setup       | 0.7.4     |
| shipping_setup         | 0.7.0     |
| sitemap_setup          | 0.7.2     |
| strikeiron_setup       | 0.9.1     |
| tag_setup              | 0.7.5     |
```

```
| tax_setup              | 0.7.11     |
| usa_setup              | 0.7.1      |
| weee_setup             | 0.13       |
| widget_setup           | 1.4.0.0.0  |
| wishlist_setup         | 0.7.7      |
+----------------------+-----+
63 rows in set (0.00 sec)
```

这张表包含了系统中所有安装的模块和模块的版本。你可以看到我们的模块

| helloworld_setup | 0.1.0 |

Magento 就是根据这个版本来判断是否需要运行升级脚本的。这里“helloworld_setup”版本“0.1.0”，而我们的安装脚本也是“0.1.0”，所以 Magento 不会再运行该脚本。如果你需要重新运行安装脚本（在开发的时候常用到），只要删除表中相应模块的数据就行了。让我们来试试看

```
DELETE from core_resource where code = 'helloworld_setup';
```

这次我们将通过安装脚本来创建数据表，所以我们也要删除之前创建的数据表

```
DROP TABLE blog_posts;
```

添加以下代码到我们的安装脚本

```
$installer = $this;
      $installer->startSetup();
      $installer->run("
          CREATE TABLE `{$installer->getTable('helloworld/blogpost')}`
(
            `blogpost_id` int(11) NOT NULL auto_increment,
            `title` text,
            `post` text,
            `date` datetime default NULL,
            `timestamp` timestamp NOT NULL default CURRENT_TIMESTAMP,
            PRIMARY KEY  (`blogpost_id`)
          ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
          INSERT INTO `{$installer->getTable('helloworld/blogpost')}`
VALUES (1,'My New Title','This is a blog post','2009-07-01
00:00:00','2009-07-02 23:12:30');
      ");
      $installer->endSetup();
```

清空 Magento 缓存，访问任何 URL，你应该发现“blog_posts”表又被建立了，拥有一条数据。

解剖配置脚本

让我们来分析一下上面的代码。【译者注：作者在文中混用了“install script”，“upgrade script”和“setup script”。我在翻译的时候尽量分清。配置脚本包含了安装脚本和升级脚本。】先看第一行

```
$installer = $this;
```

这个“$this”是什么呢？每一个配置脚本都是属于某个资源配置类（Setup Resource class），比如上面我们创建的“Zhlmmc_Helloworld_Model_Setup_Mysql4_Setup”。这些脚本都是在这个资源配置类的上下文环境中运行的。所以“$this”就是指资源配置类的实例。虽然不是强制的，但是大多数核心模块的资源配置类都用

“$installer”来代替“$this”，就和我们这里做的一样。虽然我们说不是强制的，但是我们最好还是遵守这个约定，除非你有一个很好的理由。
接下来看下面两个调用

```
$installer->startSetup();
        //…
        $installer->endSetup();
```

如果你打开“Mage_Core_Model_Resource_Setup”类（也就是我们创建的资源配置类的父类）的源码，你将会看到这两个方法做了一些 SQL 准备工作

```
public function startSetup()
        {
            $this->_conn->multi_query("SET SQL_MODE='';
        SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
        SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO';
        ");
            return $this;
        }
        public function endSetup()
        {
            $this->_conn->multi_query("
        SET SQL_MODE=IFNULL(@OLD_SQL_MODE,'');
        SET FOREIGN_KEY_CHECKS=IFNULL(@OLD_FOREIGN_KEY_CHECKS,0);
        ");
            return $this;
        }
```

真正的配置逻辑是在“run”方法中

```
$installer->run(…);
```

这个方法的参数就是你要运行的 SQL。你可以包含任意数量的 SQL，用分号隔开。你可能注意到了以下语句

```
$installer->getTable('helloworld/blogpost')
```

【译者注：你的第一反应是什么？是不是联想到 config.xml 里面下面这段代码？

```
<helloworld_mysql4>
            <class>Zhlmmc_Helloworld_Model_Resource_Mysql4</class>
            <entities>
                <blogpost>
                    <table>blog_posts</table>
                </blogpost>
            </entities>
        </helloworld_mysql4>
```

这里“helloworld/blogpost”就是我们要创建的数据表的 URI。Magento 先用“helloworld”找到模块，得到资源模块“helloworld_mysql4”，然后在资源模块下面通过“blogpost”找到类名“blog_posts”。】很显然，你可以用常量“blog_posts”来代替调用“getTable”，但是调用这个方法可以保证即使用户更改了配置文件中的数据表名字，

你的配置脚本依然能运行。“Mage_Core_Model_Resource_Setup”类有很多类似这样的帮助函数。学习这些函数最好的方法就是阅读 Magento 核心模块的资源配置类代码。

模块升级脚本

上面我们说过配置脚本包括安装脚本和升级脚本。讲完了安装脚本，我们来讲升级脚本。安装脚本是在第一次安装模块的时候使用的，而升级脚本顾名思义就是在升级模块的时候使用。Magento 的资源配置系统使用了版本化的方式来升级模块。

首先要说明的是，在安装模块的时候 Magento 会执行一次安装脚本，然后 Magento 再也不会为该模块执行任何安装脚本了。如果你要更新模块的数据表就要通过升级脚本来执行。除了命名方式以外，升级脚本和安装脚本并没有太大的不同。

创建升级脚本如下

```
File:
app/code/local/Zhlmmc/Helloworld/sql/helloworld_setup/mysql4-upgrade
-0.1.0-0.2.0.php
      echo 'Testing our upgrade script (mysql4-upgrade-0.1.0-0.2.0.php)
and halting execution to avoid updating the system version number  ';
      die();
```

升级脚本和安装脚本是放在相同目录下面的，但是命名方式不同。首先是“upgrade”关键词。其次，你会发现这里我们有两个版本号，用“-”分开。第一个版本号“0.1.0”是指从哪个版本升级（起始版本），第二个版本号“0.2.0”是指要升级到哪个版本（目标版本）。

清空 Magento 缓存，请求任何一个 URL，你会发现没有任何配置脚本被运行。那是因为，第一，我们已经运行过安装脚本，第二，目前我们模块的版本是“0.1.0”，所以 Magento 不会运行我们要升级到“0.2.0”的升级脚本。要让 Magento 运行这个升级脚本，我们得修改配置文件中的版本号

```
<modules>
          <zhlmmc_helloworld>
             <version>0.2.0</version>
          </zhlmmc_helloworld>
      </modules>
```

清空 Magento 缓存，重新请求页面，你会看到升级脚本的输出。【译者注：作者在这里引入了第二个升级到“0.1.5”的升级脚本，我觉得并没有必要，我来直接总结一下 Magento 升级的步骤

1. 从数据表“core_resource”中获得当前模块的安装版本
2. 从配置文件中获得当前模块的版本
3. 如果两个版本一样，那么什么都不做
4. 如果#2 的版本号小于#1 的版本号，我也不知道 Magento 会干什么，理论上是不可能出现的情况
5. 如果#2 的版本号大于#1 的版本号，那么开始升级程序
6. 在配置脚本文件夹内（在上面的例子中是“helloworld_setup”）把所有升级脚本加入队列
7. 在队列内，按照升级脚本的起始版本排序，升序
8. 循环队列
9. 如果队列中当前脚本的起始版本不等于“core_resource”数据表中当前模块的版本号，那么跳过该脚本
10. 如果队列中当前脚本的起始版本等于“core_resource”数据表中当前模块的版本号并且目标版本小于等于#2 的版本号，那么执行该脚本。
11. 循环队列结束，升级结束

值得注意的是第 10 步，每次执行一个升级脚本，“core_resource”数据表中的版本号都会被更新。所以如果我们有两个升级文件“0.1.0-0.1.5”和“0.1.0-0.2.0”，只有一个升级文件会被执行。

】

下面我们来为升级脚本添加实质内容

```
$installer = $this;
$installer->startSetup();
$installer->run("
    ALTER TABLE `{$installer->getTable('helloworld/blogpost')}`
    CHANGE post post text not null;
");
$installer->endSetup();
```

清空 Magento 缓存，刷新页面，你应该看到升级脚本的输出。如果没有，请你对照上面讲的 Magento 升级的步骤查找错误。

总结

这一章我们讲解了 Magento 是如何处理模块数据表的安装和升级的。Magento 的资源配置系统使用版本化的升级脚本，这样就保证了不同版本的模块可以使用同一套的升级脚本，便于维护。我们后面的章节也会提到这种升级方式的优点，特别是对于使用 EAV 模型的模块来说，这个优点更为明显。

深入理解 Magento – 第六章 – 高级 Magento 模型

我们讲过 Magento 有两种模型，简单模型和 EAV（Entity Attribute Value）模型。上一章我们讲过所有的 Magento 模型都是继承自 Mage_Core_Model_Abstract / Varien_Object。简单模型和 EAV 模型的区别在于资源模型（Model Resource）。虽然所有的资源模型都最终继承“Mage_Core_Model_Resrouce_Abstract”，但是简单模型是直接继承“Mage_Core_Model_Mysql4_Abstract”，而 EAV 模型是直接继承“Mage_Eav_Model_Entity_Abstract”。

Magento 这么做是由它的道理的。对于大部分开发人员或者用户来说，他们只需要知道一系列的方法能够操作模型，获得数据，数据到底是如何存储的并不是很重要。

什么是 EAV 模型？

Wikipedia 是这么定义的：

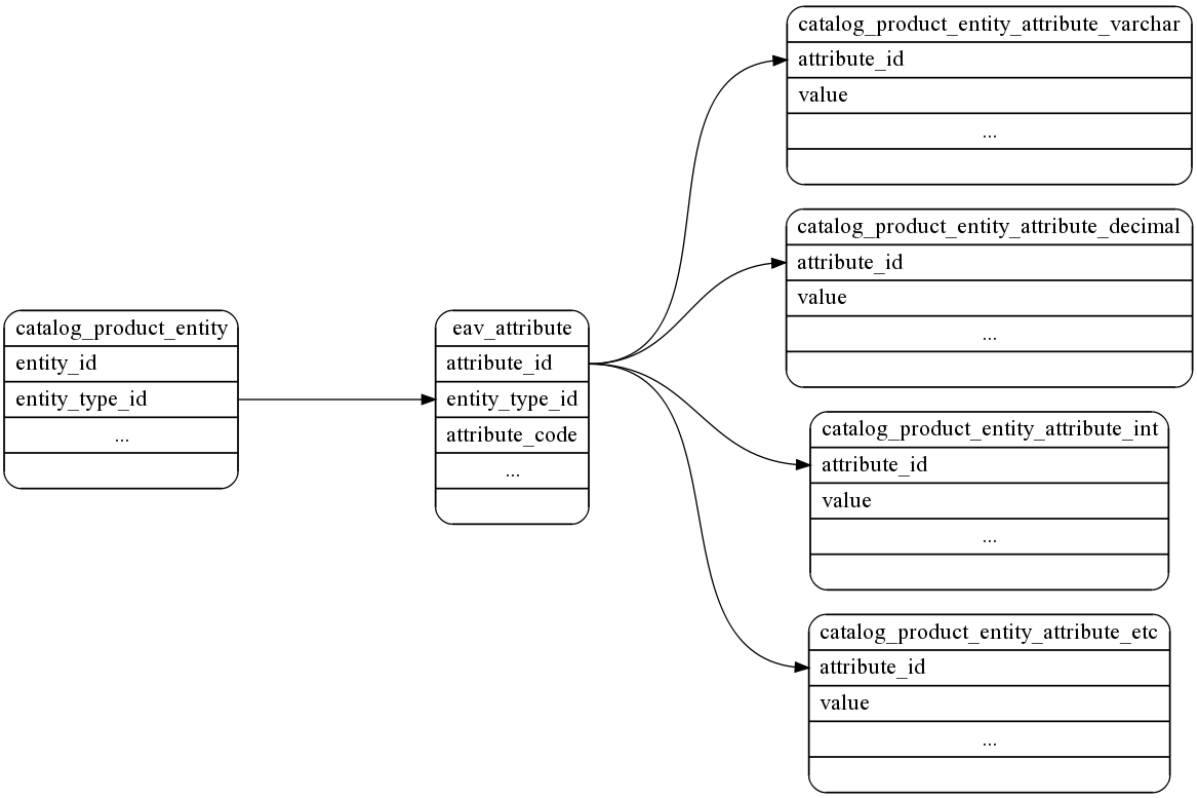
EAV（Entity-Attribute-Value）模型，也作 Object-Attribute-Value 模型或者开放模型是一种数据模型。这种数据模型常常用在一个对象的属性数目不是一定的情况下。在数学上，这种模型称为松散矩阵。

换一种方式理解，EAV 模型就是数据表的一种泛化。在传统的数据库中，数据表的列的数量是一定的

```
+--------+
| products        |
+--------+
| product_id      |
| name            |
| price           |
| etc..           |
+--------+
+------+-------+--------+----+
| product_id | name          | price           | etc…  |
+------+-------+--------+----+
| 1          | Widget A      | 11.34           | etc…  |
+------+-------+--------+----+
| 2          | Dongle B      | 6.34            | etc…  |
+------+-------+--------+----+
```

在上面这张表中，每一个商品都有名称，价格等等。

在 EAV 模型中，每一个模型都有不同的属性。这对于电子商务的应用来说是很合适的。比如说一个网店可以卖笔记本，拥有 CPU 速度，颜色，内存等属性，但是网店也可以卖衣服，有颜色属性，但是没有 CPU 速度。即使是卖衣服的网店，也有上衣和裤子之分，它们的属性也是不一样的。
有很多开源的或者商业的数据库是默认使用 EAV 模型的。但是一般的网站托管平台不提供这些数据库。所以 Varien 开发了一套基于 PHP 和 MySQL 的 EAV 系统。换句话说，它们在传统的关系型数据库上面开发了一套 EAV 数据库系统。
在使用的时候，EAV 模型的属性是会分布在不同的 MySQL 数据表中。



上面的这张图是 Magento 中关于“catalog_product”的表。每一个产品都是“catalog_product_entity”中的一行。Magento 系统中所有的属性（不仅仅是商品）都存放在“eav_attribute”表中，而属性的值都放在类似下面的表中“catalog_product_entity_attribute_varchar”， “catalog_product_entity_attribute_decimal”，“catalog_product_entity_attribute_etc”。【译者注：如果你仔细观察上面这幅数据表结构图，你会发现明显少了一张表，和“entity_type”有关。因为这里有“entity_type_id”出现，但却没有定义这个属性的表。这个表在 Magneto 中叫做“eav_entity_type”。由于 EAV 模型中所有的模型数据都混在一套数据表中了，实体类型（entity_type）就是用来把不同的模型区别开来的属性。假如我们要找出系统中所有的产品数据，那么 Magento 先通过“eav_entity_type”表获得产品模型的“entity_type_id”，然后再通过上面这幅图的关系来拿到所有的数据。
】
在 EAV 系统下面，当你需要添加一个属性的时候，只需要在“eav_attribute”表中添加一行就行了。而传统的关系型数据库则需要修改数据表调用“ALTER TABLE”语句，复杂而且有风险。EAV 模型的缺点是你不能通过一个简单的 SQL 语句就获得一个模型的所有属性。你往往需要调用多个 SQL 或者一个 SQL 包干了多个 join 语句。

实战 EAV 模型

我们已经介绍了 EAV 是怎么工作的了。下面我们要通过一个例子来说明要在 Magento 中创建一个 EAV 模型所需要的步骤。这部分内容大概是 Magento 中最令人头疼的部分，95%的 Magento 用户都不会和这些代码打交道，但是理解 EAV 模型的原理能够帮助你更好的理解 Magento 的代码和架构。
因为 EAV 模型的内容太多了，所以我假设你已经熟悉了前几章的内容，包括 Magento MVC，组类名等等。在这一章我不会再重复这些内容。

EAV 形式的 Hello World

我们将为 Hello World 模块创建另外一个模型，使用 EAV 形式的资源模型。首先我们为模块创建一个新的模型叫做“Eavblogpost”。记住，简单模型和 EAV 模型的区别是资源模型，所以我们创建一个模型的基本步骤是一样的。

```
<global>
    < !- … ->
    <models>
        < !- … ->
        <helloworld-eav>
            <class>Wemvc_Helloworld_Model</class>
            <resourceModel>helloworld-eav_mysql4</resourceModel>
        </helloworld-eav>
        < !- … ->
    </models>
    < !- … ->
</global>
```

我想我不说你也应该知道，我们要创建一个新的模型文件。由于 PHP 5.3 和命名空间（namespaces）还没有被广泛采用，Magento 中的类名仍然和文件的物理路径相关。这就导致了很多时候不知道一个 URI 所对应的类究竟该放在什么文件夹下面。我发现我们可以利用 Magento 的异常信息来直接得到一个类的路径。比如，这里我们先不创建模型类，先来修改 BlogController 来直接使用模型类，这样 Magento 就会报错说找不到模型类，并给出路径

```
public function eavReadAction(){
        $eavModel = Mage::getModel('helloworld/eavblogpost');
        echo get_class($eavModel)."<br />";
    }
```

清空 Magento 缓存，访问以下 URL

```
http://127.0.0.1/Magento/helloworld/blog/eavRead
```

跟预计的一样，你应该得到以下异常

```
Warning: include(Wemvc/Helloworld/Model/Eavblogpost.php)
[function.include]: failed to open stream: No such file or directory
```

所以我们应该创建如下文件

```
File: app/code/local/Wemvc/Helloworld/Model/Eavblogpost.php
    class Wemvc_Helloworld_Model_Eavblogpost extends
Mage_Core_Model_Abstract
    {
        protected function _construct()
        {
            $this->_init('helloworld-eav/blogpost');
        }
    }
```

刷新页面，你应该看到下面的输出

```
Wemvc_Helloworld_Model_Eavblogpost
```

下面我们来创建资源模型。先定义资源模型

```
            <helloworld-eav_mysql4>

<class>Wemvc_Helloworld_Model_Resource_Eav_Mysql4</class>
                    <entities>
                        <blogpost>
                            <table>eavblog_posts</table>
                        </blogpost>
                    </entities>
            </helloworld-eav_mysql4>
```

这里的标签名字和我们上面定义的模型的是一致的。的定义和上一章是一样的。下面的适配器的定义

```
<resources>
          < !- … ->
                <helloworld-eav_write>
                    <connection>
                        <use>default_write</use>
                    </connection>
                </helloworld-eav_write>
                <helloworld-eav_read>
                    <connection>
                        <use>default_read</use>
                    </connection>
                </helloworld-eav_read>
      </resources>
```

然后再次利用 Magento 的异常，先修改“eavReadAction”

```
public function eavReadAction(){
          $eavModel = Mage::getModel('helloworld-eav/eavblogpost');
          $params = $this->getRequest()->getParams();
          echo("Loading the blogpost with an ID of ".$params['id']."<br
/>");
          $eavModel->load($params['id']);
          $data = $eavModel->getData();
          var_dump($data);
      }
```

清空 Magento 缓存，访问 URL

```
http://127.0.0.1/Magento/helloworld/blog/eavRead/id/1
```

你应该看到如下异常

```
Warning:
include(Wemvc/Helloworld/Model/Resource/Eav/Mysql4/Blogpost.php)
[function.include]: failed to open stream: No such file or directory
```

所以我们创建相应的资源模型类

```
File:
app/code/local/Wemvc/Helloworld/Model/Resource/Eav/Mysql4/Blogpost.p
hp
      class Wemvc_Helloworld_Model_Resource_Eav_Mysql4_Blogpost
extends Mage_Eav_Model_Entity_Abstract
      {
          public function _construct()
          {
              $resource = Mage::getSingleton('core/resource');
              $this->setType('helloworld_eavblogpost');
              $this->setConnection(
                  $resource->getConnection('helloworld-eav_read'),
                  $resource->getConnection('helloworld-eav_write')
                   );
          }
      }
```

这个类和简单的资源模型就不一样。首先，我们这里继承的是“Mage_Eav_Model_Entity_Abstract”。其次，我们没有调用“_init”方法。在 EAV 模型中我们需要自己来完成资源模型初始化的过程，包括，告诉资源模型使用哪个适配器，以及实体类型（entity_type）。刷新 URL，你应该看到如下异常

```
Invalid entity_type specified: helloworld_eavblogpost
```

根据我们上文所讲的内容，那这个异常的原因很明显，那就是“eav_entity_type”表中，没有需要的“helloworld_eavblogpost”的数据。这里的“helloworld_eavblogpost”就是我们“setType”的参数。让我们来看一下这张表长什么样

```
mysql> select * from eav_entity_type\G
      *************************** 1. row ***************************
                 entity_type_id: 1
               entity_type_code: customer
                  entity_model: customer/customer
                attribute_model:
                  entity_table: customer/entity
             value_table_prefix:
                entity_id_field:
                is_data_sharing: 1
              data_sharing_key: default
         default_attribute_set_id: 1
                increment_model: eav/entity_increment_numeric
           increment_per_store: 0
          increment_pad_length: 8
           increment_pad_char: 0
```

```
 additional_attribute_table: customer/eav_attribute
entity_attribute_collection: customer/attribute_collection
*************************** 2. row ***************************
          entity_type_id: 2
        entity_type_code: customer_address
           entity_model: customer/customer_address
         attribute_model:
           entity_table: customer/address_entity
      value_table_prefix:
         entity_id_field:
         is_data_sharing: 1
       data_sharing_key: default
default_attribute_set_id: 2
        increment_model:
     increment_per_store: 0
    increment_pad_length: 8
      increment_pad_char: 0
 additional_attribute_table: customer/eav_attribute
entity_attribute_collection: customer/attribute_collection
```

正如我们前面讲过的，这张表包含了所有系统中的实体类型。我们的参数“helloworld_eavblogpost”就是实体类型的值，对应数据表列“entity_type_code”。

系统和应用程序

这一章讲的内容是 Magento 最重要的一个概念，也是很多人觉得头疼的概念。拿电脑来做比方。操作系统，比如 Mac OS X，Windows，Linux 等等，是软件系统，而浏览器，比如 FIrefox，Safari，IE 等等是应用程序。Magento 首先是一个系统，其次才是一个应用程序。你可以在 Magento 系统之上创建一个电子商务应用。令人感到困惑的是 Magento 的代码在很多地方是以很原始的方式暴露给应用程序的。EAV 系统的配置和你网店的数据存放在统一数据库中就是一个例子。

随着你越来越深入 Magento，你需要把 Magento 当作老式的 IBM 650 机器。也就是说，你必须对 Magento 有很深的了解才能对它运用自如。【译者注：这一段和上下文没什么关系，大概是作者有感而发】

创建资源配置

从理论上讲，你可以手动的在数据库中插入数据，让我们的 EAV 模型工作，但我还是不建议你这么做。所幸的是，Magento 提供了一个特殊的资源配置类，包含了一些有用的方法能自动的创建一些数据，使得系统能工作。

我们先添加资源配置

```
<resources>
        < !- … ->
                <helloworld-eav_setup>
                    <setup>
                        <module>Wemvc_Helloworld</module>

<class>Wemvc_Helloworld_Model_Entity_Setup</class>
                    </setup>
                    <connection>
                        <use>core_setup</use>
                    </connection>
```

```
                </helloworld-eav_setup>
          < !- … ->
        </resources>
```

创建资源配置类文件

```
File: app/code/local/Wemvc/Helloworld/Model/Entity/Setup.php
      class Wemvc_Helloworld_Model_Entity_Setup extends
Mage_Eav_Model_Entity_Setup {
      }
```

请注意，这里我们继承的父类是“Mage_Eav_Model_Entity_Setup”。最后，我们来创建安装脚本。如果你不熟悉这部分内容，请你参考前面章节的内容。

```
File:
app/code/local/Wemvc/Helloworld/sql/helloworld-eav_setup/mysql4-inst
all-0.1.0.php
      < ?php
      $installer = $this;
      throw new Exception("This is an exception to stop the installer
from completing");
      ?>
```

清空 Magento 缓存，访问任何页面，你应该看到以上异常。如果你没有看到异常，那说明你哪里配置错了。请注意：我们将一步一步的创建安装脚本。如果你阅读了前面的章节，你应该知道我们必须删除“core_resource”数据表中的相应数据才能使得安装脚本重新运行。所以在我们下面的例子中，当我们修改了安装脚本，我们都默认会删除“core_resource”表中的数据。正常使用 Magento 的时候我们不需要这样做的，教程中的例子是极端情况。

添加实体类型

首先我们修改安装脚本如下

```
$installer = $this;
      $installer->addEntityType('helloworld_eavblogpost',Array(
      //entity_mode is the URL you'd pass into a Mage::getModel() call
      'entity_model'        =>'helloworld-eav/eavblogpost',
      //blank for now
      'attribute_model'      =>'',
      //table refers to the resource URI helloworld-eav/blogpost
      //<helloworld
-eav_mysql4>…<blogpost><table>eavblog_posts</table>
      'table'        =>'helloworld-eav/blogpost',
      //blank for now, but can also be eav/entity_increment_numeric
      'increment_model'      =>'',
      //appears that this needs to be/can be above "1" if we're using
eav/entity_increment_numeric
      'increment_per_store'   =>'0'
      ));
```

我们调用了资源配置对象的“addEntityType”方法。这个方法的参数是实体类型（helloworld_eavblogpost）还有和这个类型相关的参数。当你运行这个脚本以后，你会发现“eav_attribute_group”，“eav_attributeset”还有“eav_entity_type”数据表中有了新的数据。访问以下 URL

```
http://127.0.0.1/Magento/helloworld/blog/eavRead/id/1
```

你应该看到以下异常

```
SQLSTATE[42S02]: Base table or view not found: 1146 Table 'eavblog_posts'
doesn't exist
```

创建数据表

我们已经告诉 Magento 我们的实体类型。接下来，我们要创建用来存储数据的数据表，并配置系统让 Magento 知道我们要用这些表。

如果你研究过 Magento 核心模块的资源配置脚本的话，比如 core/Mage/CatalogInventory 的配置脚本，你会看到很多用来创建数据表的 SQL 语句。所幸的是，我们已经不必要这样做了。Magento 提供的资源配置类有一个方法“createEntityTables”。我们可以用这个方法来创建我们需要的数据表。同时这个方法也会在 Magento 的系统数据表中添加相应的配置数据。

```
$installer->createEntityTables(
          $this->getTable('helloworld-eav/blogpost')
      );
```

“createEntityTables”有两个参数。第一个参数是基础表名（base table name）。第二个参数是一系列选项。我们这里忽略了第二个参数，这些参数都是一些高级配置，超出了我们讨论的范围。在运行了上述脚本以后，你会发现数据库中添加了如下数据表

```
eavblog_posts
      eavblog_posts_datetime
      eavblog_posts_decimal
      eavblog_posts_int
      eavblog_posts_text
      eavblog_posts_varchar
```

同时，你会发现在“eav_attribute_set”表中多了一条数据

```
mysql> select * from eav_attribute_set order by attribute_set_id DESC
LIMIT 1 \G
      *************************** 1. row ***************************
        attribute_set_id: 63
          entity_type_id: 31
      attribute_set_name: Default
              sort_order: 3
```

清空 Magento 缓存，重新访问如下 URL

```
http://127.0.0.1/Magento/helloworld/blog/eavRead/id/1
```

你应该看到以下输出

```
Loading the blogpost with an ID of 1
      array(0) { }
```

添加属性

创建资源配置的最后一步是告诉 Magento 我们的模型有哪些属性。这就和为单独的数据表添加列是一样的。【注：我们上面的输出是空的就是因为我们虽然创建了 EAV 数据表，但是却没有创建 EAV 属性，就像创建了一张没有任何列的数据表，当然是空的。】和上面的步骤一样，Magento 的资源配置类提供了相应的帮助函数，“installEntities”和“getDefaultEntities”。

我们之前所做的是告诉 Magento，我们创建了一个实体类型（Entity Type），而现在，我们要配置这个实体类型使它能够和我们的模型相符合。这个方法名字有点搞“installEntities”，其实我们要做的是配置这个实体。

修改类“Wemvc_Helloworld_Model_Setup_Entity_Setup”

```
class Wemvc_Helloworld_Model_Setup_Entity_Setup extends
Mage_Eav_Model_Entity_Setup {
<?php
class Wemvc_Helloworld_Model_Entity_Setup extends
Mage_Eav_Model_Entity_Setup {
    public function getDefaultEntities()
    {
        return array (
            'helloworld_eavblogpost' => array(
                'entity_model'     => 'helloworld-eav/eavblogpost',
                'attribute_model'   => '',
                'table'           => 'helloworld-eav/blogpost',
                'attributes'       => array(
                    'title' => array(
                        //the EAV attribute type, NOT a mysql varchar
                        'type'            => 'varchar',
                        'backend'          => '',
                        'frontend'         => '',
                        'label'           => 'Title',
                        'input'           => 'text',
                        'class'           => '',
                        'source'          => '',
                        // store scope == 0
                        // global scope == 1
                        // website scope == 2
                        'global'          => 0,
                        'visible'          => true,
                        'required'         => true,
                        'user_defined'      => true,
                        'default'          => '',
                        'searchable'       => false,
                        'filterable'       => false,
                        'comparable'       => false,
                        'visible_on_front'  => false,
                        'unique'          => false,
                    ),
                    'post' => array(
                        //the EAV attribute type, NOT a mysql varchar
```

```
                    'type'              => 'text',
                    'backend'           => '',
                    'frontend'          => '',
                    'label'             => 'Post',
                    'input'             => 'text',
                    'class'             => '',
                    'source'            => '',
                    // store scope == 0
                    // global scope == 1
                    // website scope == 2
                    'global'            => 0,
                    'visible'           => true,
                    'required'          => true,
                    'user_defined'      => true,
                    'default'           => '',
                    'searchable'        => false,
                    'filterable'        => false,
                    'comparable'        => false,
                    'visible_on_front'  => false,
                    'unique'            => false,
                ),
            ),
        )
    );
  }
}
```

这里我们构建了一个数组，数组的元素是“key/value”对。“key”就是实体类型的名字（以下代码参数是一样的“$installer->addEntityType('helloworld_eavblogpost',...)”），“value”是一个数组，用来描述这个实体类型。“value”数组的元素大部分你应该都见过，就不多解释了。这里要关注的是“attribute”元素，这个元素的值又是一个数组。这个数组的内容就是我们定义的实体类型的属性，相当于普通数据表的列，比如这里的“title”。很可惜，我无法完整解释用来描述一个属性的数组的内容。在这里，我们只要知道“type”就是这个属性的数据类型“varchar”。也就是说，这个属性的值将会被保存到“eavblog_posts_varchar”数据表中。其他的很多元素都是和 Magento 的后台管理有关。Magento 很多地方的 UI 是由模型控制的，很多这些参数都是用来控制 UI 显示和系统设置。这样做的优点是灵活性提高，但是缺点是这些内容对于外部开发者都是不透明的。【译者注：我们是可以在这个函数中返回多个实体类型的。如果返回多个实体类型，那就说明模块拥有多个模型。】顺便说一下，Magento 选择使用数组嵌套数组的形式来表示实体类型的属性很奇怪。因为 Magento 整个架构是非常面向对象的。这里的数据结构和系统的其他部分很不一样。

接下来我们需要修改安装脚本，添加如下代码

```
$installer->installEntities();
```

“installEntities”会调用“getDefaultEntities”方法来获取将要被配置的属性。当然你也可以把属性直接作为参数传给“installEntities”，但是我觉得还是按照 Magento 的习惯来比较好。在调用“installEntitis”以后，Magento 会做下面两件事

1. 在“eav_attribute”表中添加“title”属性
2. 在“eav_entity_attribute”表中添加一行

清空 Magento 缓存，刷新页面，你应该看到如下异常

```
SQLSTATE[23000]: Integrity constraint violation: 1217 Cannot delete or
update a parent row: a foreign key constraint fails
```

那是因为我们之前已经调用过一次“createEntityTables”，再次调用的时候 Magento 会尝试先删除数据表，然后再创建。但是删除的时候 Magento 没有考虑到外键的关系，先尝试删除了主表，所以就有了以上异常。为了简化教程的例子，我们暂时把“createEntityTables”语句删了。再次刷新页面，你应该看到正常的输出。

给 EAV 模型添加数据

到这里为止，我们的 EAV 模型已经创建好了，下面我们来为模型添加一些数据。在 BlogController 中添加以下方法

```
public function eavPopulateEntriesAction() {
          for($i=0;$i&lt;10;$i++) {
              $weblog2 = Mage::getModel('helloworld-eav/eavblogpost');
              $weblog2->setTitle('This is a test '.$i);
              $weblog2->save();
          }
          echo 'Done';
      }
      public function eavShowcollectionAction() {
          $weblog2 = Mage::getModel('helloworld-eav/eavblogpost');
          $entries =
$weblog2->getCollection()->addAttributeToSelect('title');
          $entries->load();
          foreach($entries as $entry)
          {
              // var_dump($entry->getData());
              echo '<h1>'.$entry->getTitle().'</h1>';
          }
          echo '<br />Done<br />';
      }
```

记得添加模型集合

```
class Wemvc_Helloworld_Model_Resource_Eav_Mysql4_Blogpost_Collection
extends Mage_Eav_Model_Entity_Collection_Abstract
      {
          protected function _construct()
          {
              $this->_init('helloworld-eav/eavblogpost',
'helloworld-eav/blogpost');
          }
      }
```

访问以下 URL

```
http://127.0.0.1/Magento/helloworld/blog/eavPopulateEntries
```

你应该看到正确的输出。细心一点的话你应该发现这里有两点比较特殊。第一，“$weblog2->getCollection()->addAttributeToSelect('title')”，这里的“title”是干什么的？因为 EAV 模型在数据库层面比较复杂，一个简单的查询都需要好多个 SQL 才能完成。所以在查询的时候你需要指明你想找什么，这样可以节省系统资源。不过你也可以传入“*”，表示查找所有数据。第二，为什么“$this->_init”有两个参数？在我们以前的章节中，简单模型的模型集合初始化的时候只需要传入模型的 URI 就可以了，为什么这里要两个参数呢？其实如果你仔细看了模型集合抽象类的代码的话，你会发现这样一段

```
if (is_null($resourceModel)) {
            $resourceModel = $model;
        }
```

所以其实是需要模型的 URI 和资源模型的 URI，但是由于我们前面章节的例子，这两个 URI 是一样的，所以省略了第二个参数。而这里，资源模型的 URI 和模型的 URI 是不一样的，所以不能省略。

总结

到这里，你应该对 Magento 整个系统的运作有所了解了。起码下一次你看到网店里面的某个商品部显示了，或者什么属性不对了，你知道去哪里找问题。除了本章介绍的内容以外 EAV 模型还有很多东西可以学习。下面是我打算在以后的文章中介绍的一些内容

1. EAV 属性：EAV 模型的属性类型不局限于 datatime, decimal, int, text 和 varchar。你可以创建自定义的数据类型。
2. 集合筛选：对 EAV 模型的数据进行筛选不是看起来的那么简单，特别是当属性是自定义类型的情况下，我们需要在集合装载之前调用“addAttributeToFilter”方法。
3. Magento EAV 模型继承：Magento 在基本的 EAV 模型之上又创建了模型的继承关系，这些继承关系可以和网店的功能直接相关，也可以优化 EAV 模型的查询。

毫无疑问，EAV 模型是 Magento 系统中最复杂的部分。不过你要始终相信一点，不管多复杂，它也就是程序。从哲学角度来讲，任何事物的产生都有特定的理由，你只需要搞清楚为什么。

深入理解 Magento – 第七章 – 自定义 Magento 系统配置

Magento 拥有十分强大的后台管理系统。作为一名开发人员，这套后台管理系统可以让你的用户简单直接的配置 Magento 系统或者你创建的模块。和 Magento 的其他功能一样，你第一次使用这套管理系统的时候可能觉得很麻烦，但是一旦你上手了，你会发现它强大的功能是那么吸引人。那么让我们开始吧。我们这一章的例子依然是基于 Helloworld 模块。

添加系统配置文件

首先我们要为模块添加一个系统配置文件。这个文件和“config.xml”是不搭界的

```
app/code/local/Zhlmmc/Helloworld/etc/system.xml
```

和全局配置（global config）相似，系统配置也是单独存储的。我们可以通过下面这段代码来获取系统配置文件

```
//header('Content-Type: text/xml');
        header('Content-Type: text/plain');
        echo $config = Mage::getConfig()
        ->loadModulesConfiguration('system.xml')
        ->getNode()
        ->asXML();
        exit;
```

你可以把这段代码放到任何执行函数（Action Method）中。“loadModulesConfiguration”方法会搜索所有配置好的模块的“etc”文件夹，寻找以传入的参数为名字的文件，在这个例子中是“system.xml”。Magento 有很

多不同的配置文件，比如 api.xml， wsdl.xml， wsdl2.xml， convert.xml， compilation.xml， install.xml。你可以为你创建的模块创建这些配置文件。

添加一个标签页

我们首先在后台系统管理页面添加一个标签页（Tab）。标签页就是后台“System->Configuration”页面左侧的导航栏。默认的标签页有 General，Catalog，Customers，Sales，Services 等等。我们来创建一个新的标签页叫做“Hello Config”。创建如下文件

```
Location: app/code/local/Zhlmmc/Helloworld/etc/system.xml
      <config>
          <tabs>
              <helloconfig translate="label" module="helloworld">
                  <label>Hello Config</label>
                  <sort_order>99999</sort_order>
              </helloconfig>
          </tabs>
      </config>
```

我们来解释一下各个节点（Tag）的意思。【译者注：由于 Tab 和 Tag 中文翻译都是标签，所以这里我把 Tag 翻译成节点，以免混淆】“”就是我们要添加的标签页的定义节点，“helloconfig”是节点的 ID。你可以任意命名这个 ID，但是必须全局唯一，也就是不能和别人用同样的 ID。这个 ID 是用来唯一标示你的标签页的。“module=helloworld”，意思是这个标签页属于哪个模块。“”节点的内容是标签的名字，也就是要显示在界面上的名字。“”指明了这个标签页显示的位置。

打开后台“System->Configuration”，你会看到如下错误

```
Fatal error: Class 'Mage_Helloworld_Helper_Data' not found in….
```

Magento Helper 简介

正如许多其他的 PHP MVC 系统一样，Magento 也有帮助类（Helper Classes）。这些类用来提供一些不适合放在模型，视图或者控制器中的功能。Magento 的帮助类也是采用分组类名的机制。也就是说我们可以覆盖默认的帮助类，同时我们需要在 config.xml 中指定帮助类的基类名。

Magento 系统默认模块有一个默认的帮助类。正如我们上面的异常显示，我们的 Helloworld 模块并没有指定一个默认的帮助类。下面让我们来添加一个。修改 config.xml

```
File: app/code/local/Zhlmmc/Helloworld/etc/config.xml
      < !- … ->
      <global>
          < !- … ->
          <helpers>
              <helloworld>
                  <class>Zhlmmc_Helloworld_Helper</class>
              </helloworld>
          </helpers>
          < !- … ->
      </global>
      < !- … ->
```

你现在应该对这类配置相当熟悉了。“”节点就是模块的名字，“”就是帮助类的基类名，命名方式如下

```
Packagename_Modulename_Helper
```

帮助类是通过全局对象 Mage 的静态方法“helper”来装载的。

```
Mage::helper('helloworld/foo')
```

根据我们的配置，上面这行代码将会装载以下类

```
app/code/local/Zhlmmc/Helper/Foo.php
      class Zhlmmc_Helloworld_Helper_Foo
```

我们上面说过 Magento 默认每个模块有一个帮助类“data”

```
Mage::helper('helloworld');
      Mage::helper('helloworld/data');
```

上面这两行代码是等价的，都会装载以下类

```
app/code/local/Zhlmmc/Helper/Data.php
      class Zhlmmc_Helloworld_Helper_Data
```

下面我们来创建我们的帮助类

```
File: app/code/local/Zhlmmc/Helper/Data.php
      class Zhlmmc_Helloworld_Helper_Data extends
Mage_Core_Helper_Abstract
      {
      }
```

清空 Magento 缓存，重新装载页面，你会发现错误不见了，但是我们的标签页还是没有出来。如果你好奇帮助类究竟能干什么，建议你去看看“Mage_Core_Helper_Abstract”类。

添加新的段

好了，帮助类的介绍到此结束。下面我们来看看为什么我们的标签页不显示出来。在 Magento 中，每一个标签页都包含很多段（section）。举个例子，“Advanced”标签页默认包含“Admin, System, Advanced, Developer”四个段。如果一个标签页不包含任何段，那么这个标签页不会被显示出来。下面我们在 system.xml 中添加“”节点

```
Location: app/code/local/Zhlmmc/Helloworld/etc/system.xml
      <config>
         <tabs>
            <helloconfig translate="label" module="helloworld">
               <label>Hello Config</label>
               <sort_order>99999</sort_order>
            </helloconfig>
         </tabs>
         <sections>
            <helloworld_options translate="label"
module="helloworld">
               <label>Hello World Config Options</label>
               <tab>helloconfig</tab>
               <frontend_type>text</frontend_type>
               <sort_order>1000</sort_order>
               <show_in_default>1</show_in_default>
```

```
                <show_in_website>1</show_in_website>
                <show_in_store>1</show_in_store>
            </helloworld_options>
        </sections>
    </config>
```

这里有些节点你应该很熟悉，就不多解释了，来讲讲以前没见过的。

什么是？

和前面的相似，这个节点是用来唯一标示你的段，“helloworld_options”就是段的 ID，可以随意取名，只要不重复就好。

什么是？

这个节点有点搞。“”在配置文件的其他部分有用（稍后会讲），放在这里其实没什么作用。但是核心模块在此处的配置文件都包含这个节点，所以我们也把它添加进去。

什么是，< show_in_website />，？

这些节点的值是布尔类型的，0 或者 1。这些标签是用来控制在不同的环境下，当前段是否应该显示。

好了，我们已经配置好段了，清空缓存，再一次刷新页面，你应该看到“HELLO CONFIG”标签页显示出来了。

访问控制

如果你刚才点了我们创建的标签页下面的“Hello World Config Options”，你大概会很失望。什么都没有显示出来，连左边的导航栏都没有了。这是因为“Adminhtml”在权限控制列表（Access Control List, ACL）中找不到我们创建的段的权限信息。【译者注：Adminhtml 就是 Magento 的后台管理系统，属于 Magento 的一个核心模块】

ACL 是一个很复杂的话题，但是我会介绍一些最基本的概念，以便于理解 Magento 的权限控制。这部分内容和上下文关系不大，如果你不感兴趣，可以直接跳到本节结尾，复制一段 XML 到你的 config.xml 就行了。

在 Magento 中，对于有些资源的访问时有限制的。用户必须先经过认证才能访问相关资源。在这里，资源（Resource）是一个广义的概念，它可能是指一个页面，也可能是一个功能。Magento 的系统配置功能（System Config）就是需要认证才能访问的资源。

任何一个资源都是通过一个 URI 来标识。比如说“web”配置段（属于后台管理 General 标签页）的 URI 是

```
admin/system/config/web
```

我们“helloworld_options”段的 URI 是

```
admin/system/config/helloworld_options
```

当一个用户访问一个受保护的资源的时候，后台管理系统（Adminhtml）的执行控制器会执行以下步骤

1. 为用户正在访问的资源生成一个 URI
2. 根据 ACL 系统检查该用户是否有权限访问指定的资源
3. 如果用户拥有访问权限，那么进行用户指定的操作。否则，跳转到相应的错误页面（也可能是停止操作或者显示空白页面）。

如果你去“System -> Permissions -> Roles”页面，点击“Add New Role”按钮，你会看到所有系统的资源都以树形结构显示在页面上。

添加 ACL 权限

刚才说 ACL 中没有我们配置段的信息，那么我们来创建一个。请注意，如果你是创建一个新的段，那么你需要创建一个新的权限，如果你在已有的段上添加内容，你不需要创建权限。

在 config.xml 中，添加以下内容

```
File: app/code/local/Zhlmmc/Helloworld/etc/config.xml
    <config>
        < !- … ->
```

```
          <adminhtml>
              <acl>
                  <resources>
                      <admin>
                          <children>
                              <system>
                                  <children>
                                      <config>
                                          <children>
                                              <helloworld_options>
                                                  <title>Store Hello World
Module Section</title>
                                              </helloworld_options>
                                          </children>
                                      </config>
                                  </children>
                              </system>
                          </children>
                      </admin>
                  </resources>
              </acl>
          </adminhtml>
          < !- … ->
      </config>
```

让我们来分析一下这段代码。所有的资源定义都包含在如下代码中

```
<adminhtml>
          <acl>
              <resources>

          </resources></acl>
      </adminhtml>
```

在节点下面，每一个子节点都是 URI 的一部分，比如

```
<admin>
          <children>
              <system>
                  <children></children></system></children></admin>
```

代表 URI

```
admin/system
```

最后一个节点

```
<helloworld_options>
          <title>Store Hello World Module Section</title>
```

```
        </helloworld_options>
```

这里

清空 Magento 缓存，刷新页面，你应该能看到我们创建的配置段了，标准的后台管理页面，但是主体内容是空的，只有一个“Save Config”按钮。你可能需要重新登录后台管理才能看到正确的页面。那是因为后台管理有一些额外的缓存。【译者注：我们添加了权限以后，管理员是默认拥有该权限的，所以我们用管理员登录后台管理系统就能访问我们创建的段】

请注意，不懂事出于什么原因，Magento 把部分从全局配置中删掉了。所以，我们不能用之前创建的 Configviewer 来查看这部分内容。我正在研究 Magento 把存在哪里了。

添加组

【译者注：按照逻辑，这里应该讲的内容是添加选项。Mageto 中，选项是按照组（Group）来划分的，所以我们在添加选项之前得先添加组。】修改 system.xml

```
Location: app/code/local/Zhlmmc/Helloworld/etc/system.xml
        <config>
            <tabs>
                <helloconfig translate="label" module="helloworld">
                    <label>Hello Config</label>
                    <sort_order>99999</sort_order>
                </helloconfig>
            </tabs>
            <sections>
                <helloworld_options translate="label"
module="helloworld">
                    <label>Hello World Config Options</label>
                    <tab>helloconfig</tab>
                    <frontend_type>text</frontend_type>
                    <sort_order>1000</sort_order>
                    <show_in_default>1</show_in_default>
                    <show_in_website>1</show_in_website>
                    <show_in_store>1</show_in_store>
                    <groups>
                        <messages translate="label">
                            <label>Demo Of Config Fields</label>
                            <frontend_type>text</frontend_type>
                            <sort_order>1</sort_order>
                            <show_in_default>1</show_in_default>
                            <show_in_website>1</show_in_website>
                            <show_in_store>1</show_in_store>
                        </messages>
                    </groups>
                </helloworld_options>
            </sections>
        </config>
```

这里也没什么好解释的。刷新一下页面看看你就什么都明白了。

添加配置选项

最后，我们要添加每一个单独的配置选项。配置选项是以节点的形式添加到节点下面的。

```
< !- … ->
      <messages translate="label">
         <label>Demo Of Config Fields</label>
         <frontend_type>text</frontend_type>
         <sort_order>1</sort_order>
         <show_in_default>1</show_in_default>
         <show_in_website>1</show_in_website>
         <show_in_store>1</show_in_store>
         <fields>
            <hello_message>
               <label>Message</label>
               <frontend_type>text</frontend_type>
               <sort_order>1</sort_order>
               <show_in_default>1</show_in_default>
               <show_in_website>1</show_in_website>
               <show_in_store>1</show_in_store>
            </hello_message>
         </fields>
      </messages>
      < !- … ->
```

这里有一个节点需要说明，“”，刚才说这个节点没什么用。但是这里有用了，这个节点说明了这个选项的数据类型。你可以把它换成别的类型，比如“time”。这里支持大部分默认的 Varien 定义的数据类型（lib/Varien/Data/Form/Element）。这个有点像是工厂（Factory）设计模式。让我们把类型改成“select”。你会看到一个下拉框，但是没有选项。我们来添加选项。首先我们要添加一个源模型（Source Model）

```
<hello_message>
         <label>Message</label>
         <frontend_type>select</frontend_type>
         < !- adding a source model ->
         <source_model>helloworld/words</source_model>
         <sort_order>1</sort_order>
         <show_in_default>1</show_in_default>
         <show_in_website>1</show_in_website>
         <show_in_store>1</show_in_store>
      </hello_message>
```

“”定义了源模型的 URI。和我们以前创建的模型一样，源模型也是一个模型，为“select”提供了默认的数据。我想我不说你也明白，根据这里的 URI 定义，我们要创建以下文件

```
File: app/code/local/Zhlmmc/Helloworld/Model/Words.php
      class Zhlmmc_Helloworld_Model_Words
      {
         public function toOptionArray()
         {
            return array(
```

```
                array('value'=>1,
'label'=>Mage::helper('helloworld')->__('Hello')),
                array('value'=>2,
'label'=>Mage::helper('helloworld')->__('Goodbye')),
                array('value'=>3,
'label'=>Mage::helper('helloworld')->__('Yes')),
                array('value'=>4,
'label'=>Mage::helper('helloworld')->__('No')),
            );
        }
    }
```

源模型提供了一个方法“toOptionsArray”，返回的数据时用来填充我们之前定义的配置选项的。这个方法在运行时会被“initFields”调用。“initFields”在以下类中定义

```
app/code/core/Mage/Adminhtml/Block/System/Config/Form.php
```

我们这里调用了帮助类的翻译函数（__）来获取数据。虽然不是很必要，但调用翻译函数总是一个好习惯。说不定哪天你要将模块翻译成日文呢。【译者注：值得注意的是我们这里创建的模型不需要继承任何父类，只需要拥有“toOptionArray”方法就可以了。我觉得这个很不科学，起码要继承一个接口吧】

在已有的配置段或者组中添加数据

除了新建一个标签页，或者配置段，你也可以利用已有的标签页和配置段，向里面添加内容。比如我们添加以下代码到 system.xml

```
File: app/code/local/Zhlmmc/Helloworld/etc/system.xml
    <config>
        < !- … ->
        <sections>
            < !- … ->
            <general>
                <groups>
                    <example>
                        <label>Example of Adding a Group</label>
                        <frontend_type>text</frontend_type>
                        <sort_order>1</sort_order>
                        <show_in_default>1</show_in_default>
                        <show_in_website>1</show_in_website>
                        <show_in_store>1</show_in_store>
                    </example>
                </groups>
            </general>
            < !- … ->

    </sections></config>
```

刷新页面，你会在“General”标签页下面看到一个新的组，叫做“Example of Adding a Group”。

如何获得配置数据

到目前为止，我们只是讲了如何设置 Magento，可以让用户可以配置我们的模块。现在让我们来看看如何获取用户的配置数据。

```
Mage::getStoreConfig('helloworld_options/messages/hello_message');
```

上面这行代码就可以获取我们上面配置的那个“select”选项的数据。这个函数的参数是我们要获取的数据的 URI，格式如下

```
section_name/group_name/field_name
```

你也可以通过以下代码来获取一个组或者段的所有值

```
Mage::getStoreConfig('helloworld_options/messages');
      Mage::getStoreConfig('helloworld_options');
```

最后，如果你想获取针对某个特定店面（store）的数据，你可以传入 store ID

```
Mage::getStoreConfig('helloworld_options',1);
```

总结

这一章我们讲了如何在 Magento 的后台管理中添加个性化的配置。我们也顺便介绍了帮助类的使用和 ACL 基础。这里最重要的内容是后台配置的层级结构，标签页包含了配置段，配置段包含了组，组包含了配置选项。我们将在以后的章节中介绍系统配置的高级内容，包括自定义格式，数据验证等等。

深入理解 Magento－第八章 – 深入自定义 Magento 系统配置(未完成)

深入理解 Magento－第九章－修改、扩展、重写 Magento 代码

作为一个开发者的你，肯定要修改 Magento 代码去适应你的业务需求，但是在很多时候我们不希望修改 Magento 的核心代码，这里有很多原因，例如将来还希望升级 Magento、还想使用更多的 Magento 代码。如果你正在寻找修改 Magento 代码的最佳方式，那么此篇文章将会是一个不错的教程。
适合对象：高级开发者
适合目标：开发者希望自定义修改 Magento
当前版本：Magento versions: 1.4.0.1
作者：精东
最后修改时间：2010 年 7 月 13 日
版本：V 0.3.0

重写 Magento 模块(Module)

第一步，你需要创建属于你自己代码的命名空间，例如 Wemvc，App 等，为了方便与大家分享代码，我将空间命名为 App。

```
app/
    code/
          core/
          community/
          local/
                App/
```

假如你现在打算修改 Mage/Catalog/Block/Breadcrumbs.php 这个文件，你可以在你的命名空间，App 里添加一个新的模块“Catalog”。接下来创建块(Block)目录，并复制 Breadcrumbs.php 到你的新目录中。这里还需要你创建一个 config.xml 配置文件。

```
app/
    code/
          core/
          community/
          local/
                App/
                        Catalog/
                                Block/
                                      Breadcrumbs.php
                                etc/
                                      config.xml
```

修改 Breadcrumbs.php 的类名为 App_Catalog_Block_Breadcrumbs，并继承原类名 Mage_Catalog_Block_Breadcrumbs。

现在，你需要激活你的新模块，这样 magento 才能够知道你的新模块。

创建文件 app/etc/modules/App_All.xml，添加如下代码。

```
< ?xml version="1.0"?>
<config>
    <modules>
       <App_Catalog>
          <active>true</active>
          <codePool>local</codePool>
       </App_Catalog>
    </modules>
</config>
```

下面我们需要一个特殊的标签来复写掉 Breadcrumbs，下面我们通过模块的配置文件来实现。

重写 Magento 区块(Blocks)

编辑文件“app/code/local/App/Catalog/etc/config.xml”

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <modules>
      <App_Catalog>
         <version>0.1.0</version>
      </App_Catalog>
   </modules>
   <global>
      <blocks>
         <catalog>
            <rewrite>

<breadcrumbs>App_Catalog_Block_Breadcrumbs</breadcrumbs>
            </rewrite>
         </catalog>
      </blocks>
   </global>
```

```
</config>
```

我们需要添加一个“blocks” 标签，或者在已经存在的“blocks”标签中添加内容。然后在模块名后面添加 rewrite 标签，在这个例子中模块名是“catalog”。然后我们看“breadcrumbs”标签，这个标签帮助 magento 找到我们我们想修改的块。在我们的列子中，breadcrumbs 是 Magento 核心代码中的类名: app/code/core/Mage/Catalog/Block/Breadcrumbs.php。如果你有更多的目录层级，可以用下滑线来分隔。例如：

```
    <blocks>
        <catalog>
            <rewrite>

<category_view>App_Catalog_Block_Category_View</category_view>
            </rewrite>
        </catalog>
    </blocks>
```

在这个例子中，我们重写了 app/code/core/Mage/Catalog/Block/Category/View.php。
在 breadcrumbs 标签中的值是你的类名，这样 Magento 就可以获取你的类，因为类名与你的目录名一致。用过 zend framework 的人都知道，自动加载 auto loader 这个东西，它会跟你类名中的下滑线去你的目录中需要对应的类文件。记住一点，下滑线代表下一级别的文件夹，如果你的类名与你的文件目录名不一致，那么 Magento 根本不会理睬你。
举例来说：

```
App_Catalog_Block_Breadcrumbs →
/app/code/local/App/Catalog/Block/Breadcrumbs.php
App_Catalog_Block_Category_View →
/app/code/local/App/Catalog/Block/Category/View.php
```

重写 Magento 控制器(Controller)-正则表达式匹配式

重写 Magento 控制器我们我们以重写购物车为例。
1、首先在 App 下创建新的模块，依次创建如下文件：

```
/app/code/local/App/Shopping
/app/code/local/App/Shopping/etc
/app/code/local/App/Shopping/etc/config.xml
/app/code/local/App/Shopping/controllers
/app/code/local/App/Shopping/controllers/CartController.php
```

2、编辑/app/code/local/App/Shopping/etc/config.xml 文件，加入如下代码：

```
<?xml version="1.0"?>
<config>
    <modules>
        <App_Shopping>
            <version>0.1.0</version>
        </App_Shopping>
    </modules>
    <global>
```

```
            <!-- This rewrite rule could be added to the database instead
-->
            <rewrite>
                  <!-- This is an identifier for your rewrite that
should be unique -->
                  <!-- THIS IS THE CLASSNAME IN YOUR OWN CONTROLLER -->
                  <App_Shopping_cart>
                        <from><![CDATA[#^/checkout/cart/#]]></from>
                        <!--
                              - Shopping module matches the router
frontname below - checkout_cart
                              matches the path to your controller
Considering the router below,
                              "/shopping/cart/" will be "translated"
to

      "/App/Shopping/controllers/CartController.php" (?)
                        -->
                        <to>/shopping/cart/</to>
                  </App_Shopping_cart>
            </rewrite>
      </global>
      <!--
            If you want to overload an admin-controller this tag should
be <admin>
            instead, or <adminhtml> if youre overloading such stuff (?)
      -->
      <frontend>
            <routers>
                  <App_Shopping>
                        <!-- should be set to "admin" when overloading
admin stuff (?) -->
                        <use>standard</use>
                        <args>
                              <module>App_Shopping</module>
                              <!-- This is used when "catching" the
rewrite above -->
                              <frontName>shopping</frontName>
                        </args>
                  </App_Shopping>
            </routers>
      </frontend>
</config>
```

3、改写你自己的控制器
/app/code/local/App/Shopping/controllers/CartController.php
请将下面的代码添加到你的控制器中，我们唯一修改的地方是在 index 动作中添加一个 error_log();

```
< ?php

# 控制器不会自动加载，所以我们需要包含文件，这里与区块(Block)不一样

require_once 'Mage/Checkout/controllers/CartController.php';
class App_Shopping_CartController extends Mage_Checkout_CartController
{

    #覆写 indexAction 方法

    public function indexAction()
    {
            # Just to make sure

            error_log('耶～成功重写购物车！');

            parent::indexAction();
    }
}
```

在这段代码中，首先是类名，跟前面讲到的区块(Block)一样，我们自己的类名是 App_Shopping_CartController 继承原先 Mage_Checkout_CartController.在 indexAction 中我们记录了一段信息。
4、修改 App_All.xml，激活我们新的 Shopping 模块

```
<?xml version="1.0"?>
<config>
    <modules>
       <App_Catalog>
          <active>true</active>
          <codePool>local</codePool>
       </App_Catalog>
       <App_Shopping>
          <active>true</active>
          <codePool>local</codePool>
       </App_Shopping>
    </modules>
</config>
```

到这里，清除缓存后，你已经可以看到 error_log 成功记录了我们的信息，打开页面 http://www.wemvc.dev/checkout/cart/，显示的是购物车页面，一切正常，但如果你访问 http://www.wemvc.dev/shopping/cart/，你会发现是首页。。。。我们期望的购物车视图还没有出现，如何解决呢？让我们接下来往下看。
5、修改视图文件 app/design/frontend/[myinterface]/[mytheme]/layout/checkout.xml
在 layout 标签中，添加下面内容：

```
       <app_shopping_cart_index>
            <update handle="checkout_cart_index"/>
```

```
        </app_shopping_cart_index>
```

注意，这里的大小写敏感。

到这里基本大功告成，但是，我建议你学习下正则表达式，因为刚刚的代码中，有这么一段：

```
<from>< ![CDATA[#^/checkout/cart/#]]></from>
```

这里是使用正则表达式进行匹配的。

还有一点，经过尝试，这里是可以支持同模块名覆盖的，例如 Magento 代码中商品详情页是 Mage_Catalog_ProductController::viewAction()，如果我们想重写这个 Controller，我们可以这样做：

a.简历新的目录/app/code/local/App/Catalog/controllers/ProductController.php

代码如下:

```
require_once 'Mage/Catalog/controllers/ProductController.php';

/**
 * Product controller
 *
 * @category   Mage
 * @package    Mage_Catalog
 */
class App_Catalog_ProductController extends
Mage_Catalog_ProductController
{
    /**
     * View product action
     */
    public function viewAction()
    {

        echo '覆盖过的....';

        parent::viewAction();
    }
}
```

b.编辑/app/code/local/App/Catalog/etc/config.xml，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <modules>
        <App_Catalog>
            <version>0.1.0</version>
        </App_Catalog>
    </modules>
    <global>
                <!-- This rewrite rule could be added to the database instead
-->
                <rewrite>
```

```
                    <!-- This is an identifier for your rewrite that
should be unique -->
                    <!-- THIS IS THE CLASSNAME IN YOUR OWN CONTROLLER -->
                    <App_Shopping_cart>

        <from><![CDATA[#^/catalog/product/#]]></from>
                            <!--
                                    - Shopping module matches the router
frontname below - checkout_cart
                                    matches the path to your controller
Considering the router below,
                                    "/shopping/cart/" will be "translated"
to

        "/App/Shopping/controllers/CartController.php" (?)
                            -->
                            <to>/catalog/product/</to>
                    </App_Shopping_cart>
                </rewrite>
        <blocks>
            <catalog>
                <rewrite>

<breadcrumbs>App_Catalog_Block_Breadcrumbs</breadcrumbs>
                </rewrite>
            </catalog>
        </blocks>
    </global>
        <frontend>
        <routers>
            <catalog>
                <use>standard</use>
                <args>
                    <module>App_Catalog</module>
                    <frontName>catalog</frontName>
                </args>
            </catalog>
        </routers>
        </frontend>
</config>
```

清空缓存，刷新你的商品详情页，看是不是变了，呵呵。但是这个方法有个弊病，你需要把这个模块的所有 Controller 都复写掉，不然你会遇到比较大的麻烦。说到这，我再介绍一种重写方法.
仔细看配置文件的写法：

```
<?xml version="1.0"?>
<config>
```

```
    <modules>
        <App_Mycms>
            <version>0.1.0</version>
        </App_Mycms>
    </modules>
    <frontend>
        <routers>
            <mycms>
                <use>standard</use>
                <args>
                    <module>App_Mycms</module>
                    <frontName>mycms</frontName>
                </args>
            </mycms>
        </routers>
    </frontend>
    <global>
        <routers>
            <cms>
                <rewrite>
                    <index>
                        <to>App_Mycms/index</to>
                        <override_actions>true</override_actions>
                        <actions>

<noroute><to>App_Mycms/index/noroute</to></noroute>
                        </actions>
                    </index>
                </rewrite>
            </cms>
        </routers>
    </global>
</config>
```

综上所述，三种重写方法都各有千秋，关键看你用在什么地方。另外我们在实践中发现，Magento 好像不建议你自己的模块名与现有系统中的模块名一致，例如 Mage_Customer 是已有的，它的模块名叫 Customer，如果你想复写它，那么最好你再建一个 App_Customers 之类的。

重写 Magento 模型和动作助手(Model&Helper)

我们在改写 Magento 的过程中，为了实现自己的业务逻辑，难免要改它的业务模型。你可以尝试用模块下的配置文件配置你自己的类，继承你想重写的模型或者助手，然后调用自己的类。现在我们以用户模型为例深入讲解。

a.首先创建自己的模块文件夹

```
app/code/local/App/Customer
app/code/local/App/Customer/etc/config.xml
app/code/local/App/Customer/Model
```

```
app/code/local/App/Customer/Model/Customer.php
```

b.修改 app/etc/modules/App_All.xml

```
        <App_Customer>
            <active>true</active>
            <codePool>local</codePool>
        </App_Customer>
```

c.修改自己的模块配置文件 app/code/local/App/Customer/etc/config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
        <modules>
                <App_Customer>
                        <version>0.1.0</version>
                </App_Customer>
        </modules>

        <global>
                <models>
                        <customer>
                                <rewrite>

        <customer>App_Customer_Model_Customer</customer>
                                </rewrite>
                        </customer>
                </models>
        </global>
</config>
```

d.现在写你新的 Model，在文件 app/code/local/App/Customer/Model/Customer.php 中新建类 App_Customer_Model_Cutomer

```
class App_Customer_Model_Customer extends Mage_Customer_Model_Customer
{

        // 重写已存在的方法

        public function validate() {
                // Define new validate rules. From now magento call this
validate method instead of existing method
                //return $errors;
                return true;
        }


        // 你还可以创建新的方法

        public function newMethod() {
                // function logic
```

```
        }
}
```

e.我们再重写一个类，以加深理解。接下来我们重写 Customer Address Model。 跟重写 Customer Model 一样，我们先编辑模块的配置文件 app/code/local/App/Customer/etc/config.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
        <modules>
                <App_Customer>
                        <version>0.1.0</version>
                </App_Customer>
        </modules>

        <global>
                <models>
                        <customer>
                                <rewrite>

        <customer>App_Customer_Model_Customer</customer>

        <address>App_Customer_Model_Address</address>
                                </rewrite>
                        </customer>
                </models>
        </global>
</config>
```

上面看出来了么，rewrite 标签内的 customer 和 address 其实就是你要覆写的 magento model。
接下来创建 model class App_Customer_Model_Address，并写你要覆盖和新增的方法

```
class App_Customer_Model_Address extends Mage_Customer_Model_Address {

        // 重写已存在的方法

        public function validate() {
                // Define new validate rules. From now magento call this
validate method instead of existing method
                //return $errors;
                return true;
        }


        // 你还可以创建新的方法

        public function newMethod() {
                // function logic
        }
}
```

f.我再讲下如何覆盖 Magento 的模型资源，这里以复写 Address Entity Model class 为例，我们先来修改模块的配置文件 app/code/local/App/Customer/etc/config.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
	<modules>
		<App_Customer>
			<version>0.1.0</version>
		</App_Customer>
	</modules>

	<global>
		<models>
			<customer>
				<rewrite>

	<customer>App_Customer_Model_Customer</customer>

	<address>App_Customer_Model_Address</address>
				</rewrite>
			</customer>
			<customer_entity>
				<rewrite>

	<address>App_Customer_Model_Entity_Address</address>
				</rewrite>
			</customer_entity>
		</models>
	</global>
</config>
```

接下来创建类文件。

```
class App_Customer_Model_Entity_Address extends
Mage_Customer_Model_Entity_Address {
	protected function _afterSave(Varien_Object $address) {
		// Write your code
	}
}
```

总结

在本章中我们学习了如何重写模块、重写控制器、重写区块，以及如何重写模型和助手，基本重写 Magento 代码对你来说已经不是难事了。文章至此，要恭喜你，你已经掌握了大部分修改 Magento 的技能。下面的文章我们会进行更深入的研究。最后感谢所有 Sasacake Team Member，是他们对待工作的热情和责任感促使我写这些教程。

深入理解 Magento－第十章－数据操作&数据收集器

在我们开始介绍数据操作前，我们先介绍一个神兵利器—Varien Data Collections。在最早的时候我们写 php 通常用 Array 来做数据收集器，这个小东西可发挥了大作用，要知道如果你想在其他语言中实现 Array 有多么难过。
例如 c、c＋＋。
在 php5 中，更是发扬了 Array，php 内置了一些类和接口，允许你创建你自己的数据结构。Magento 充分利用了这一点，在使用 Varien_Data_Collection 来做数据收集的时候，它实现了 php 内置 IteratorAggregate 对象迭代器和 Countable 两个接口。下面是用 php 内置类 ArrayObject 的一个例子。

```
$array = new ArrayObject();
class MyCollection extends ArrayObject{}
$collection = new MyCollection();
$collection[] = 'bar';
```

在接下来的文章中，我认为你已经了解 ArrayObject、IteratorAggregate、Countable。如果还是很陌生，我建议你先阅读这篇文章 PHP5 对象迭代(Object Iteration)。当然你不必了解很底层的东西，你只需要知道如何用就可以了。
适合对象：高级开发者
作者：精东
最后修改时间：2010 年 5 月 31 日
版本：V 0.1.0

在 Magento 代码中，其实每个 Model 都有个 Collection。了解这些数据收集器是如何工作的是你成为一个真正 Magento 开发人员的关键点。
下面让我们开始吧，前面我们创建过一个 Helloworld 模块，现在我们继续用他开始我们接下来的学习。
创建一个数据收集器</H4< p>
首先，我们创造一些新的对象。

```
$thing_1 = new Varien_Object();
$thing_1->setName('Richard');
$thing_1->setAge(24);

$thing_2 = new Varien_Object();
$thing_2->setName('Jane');
$thing_2->setAge(12);

$thing_3 = new Varien_Object();
$thing_3->setName('Spot');
$thing_3->setLastName('The Dog');
$thing_3->setAge(7);
```

Magento 中所有的 Model 都继承 Varien_Object，在面向对象编程中，这样做的好处是当你想往多个 Model 中添加方法的时候，你只需要简单地修改一个文件即可。
在继承 Varien_Object 的类中，有两个魔术方法，get/set，你可以很方便的向对象中加入一个属性（值），让我们看个例子。

```
var_dump($thing_1->getName());
```

如果你忘记了属性的名字，你可以将所有数据都获取到：

```
var_dump($thing_3->getData());
```

你将看到以下结果：

```
array
'name' => string 'Spot' (length=4)
'last_name' => string 'The Dog' (length=7)
'age' => int 7
```

注意 last_name 属性，是用下滑线分隔的，如果你想用 get 和 set 魔术方法，那么需要使用驼峰命名法。

```
$thing_1->setLastName('Smith');
```

在新版本的 magento 中你可以用 array 关联数组的方式获取数据。

```
var_dump($thing_3["last_name"]);
```

T 这个归功于 php5 的新特性，ArrayAccess 接口。也是 “Object Oriented Programming”.
现在然我们把这些对象加到数据收集器 Varien_Data_Collection 中。很多程序员将 Collection 看成是数组，当然我不反对。

```
$collection_of_things = new Varien_Data_Collection();
$collection_of_things
->addItem($thing_1)
->addItem($thing_2)
->addItem($thing_3);
```

大多数 Magento data Collections 继承于 Varien_Data_Collection，你可以使用里面的任何一个方法。
那么我们可以做些什么呢？接下来我们使用 foreach 去循环它。

```
foreach($collection_of_things as $thing)
{
    var_dump($thing->getData());
}
```

这里还有方法取出第一个数据和最后一个数据。

```
var_dump($collection_of_things->getFirstItem());
var_dump($collection_of_things->getLastItem()->getData());
```

将你的数据转成 xml

```
var_dump( $collection_of_things->toXml() );
```

只像取某一个字段

```
var_dump($collection_of_things->getColumnValues('name'));
```

Magneto 还给我们提供了一些基本的过滤功能

```
var_dump($collection_of_things->getItemsByColumnValue('name','Spot')
);
```

模型数据收集器（Model Collections）
前面我们有提到，所有 Magento 的模型数据收集器都继承 Varien_Data_Collectionm，所以理论上我们可以使用之前的所有方法。下面让我们以 product 模型实战下。

```
public function testAction()
{
    $collection_of_products =
Mage::getModel('catalog/product')->getCollection();
    var_dump($collection_of_products->getFirstItem()->getData());
}
```

基本所有的 Magento 模型都有个方法叫 getCollection 默认情况下，它会返回系统中所有的数据。
Magento 的数据收集器 Collection 包含很多复杂的逻辑来处理数据，无论是否使用索引或缓存、EAV 表等。
上面的产品数据收集器，它里面还有 Varien_Data_Collection_Db 类。这个类给你很多有用的方法，例如如果你向看 sql 的 select 语句。

```
public function testAction()
{
    $collection_of_products =
Mage::getModel('catalog/product')->getCollection();
    var_dump($collection_of_products->getSelect()); //might cause a
segmentation fault
}
```

上面的方法将输出

```
object(Varien_Db_Select)[94]
  protected '_bind' =>
    array
      empty
  protected '_adapter' =>
...
```

从上面可以看出，Magento 使用的是 ZendFramework 的数据库链接层。接下来让我们看看更有意义的东西

```
public function testAction()
{
    $collection_of_products =
Mage::getModel('catalog/product')->getCollection();
    //var_dump($collection_of_products->getSelect()); //might cause a
segmentation fault
    var_dump(
        (string) $collection_of_products->getSelect()
    );
```

```
}
```

上面的方法将输出

```
'SELECT `e`.* FROM `catalog_product_entity` AS `e`'
```

有时也会比较复杂，例如

```
string 'SELECT `e`.*, `price_index`.`price`,
`price_index`.`final_price`, IF(`price_index`.`tier_price`,
LEAST(`price_index`.`min_price`, `price_index`.`tier_price`),
`price_index`.`min_price`) AS `minimal_price`,
`price_index`.`min_price`, `price_index`.`max_price`,
`price_index`.`tier_price` FROM `catalog_product_entity` AS `e`
INNER JOIN `catalog_product_index_price` AS `price_index` ON
price_index.entity_id = e.entity_id AND price_index.website_id = '1' AND
price_index.customer_group_id = 0'
```

这个差异取决于你选择的字段，同样也涉及到索引和缓存。如果你看过之前的文章，那么你应该知道很多 Magento 表是使用 Eav 表结构的，默认情况下一个 eav 的数据收集器将不会包含所有的对象字段，你可以通过 addAttributeToSelect 来添加它们。让我们看看例子。

```
$collection_of_products = Mage::getModel('catalog/product')
->getCollection()
->addAttributeToSelect('*');  //the asterisk is like a SQL SELECT *
```

或者你也可以只选某一个字段

```
//or just one
$collection_of_products = Mage::getModel('catalog/product')
->getCollection()
->addAttributeToSelect('meta_title');
```

或者更多

```
//or just one
$collection_of_products = Mage::getModel('catalog/product')
->getCollection()
->addAttributeToSelect('meta_title')
->addAttributeToSelect('price');
```

延迟加载（Lazy Loading）

一般情况下，我们在创建 sql 后需要立刻执行，从而获取数据，例如。

```
$model = new Customer();
//SQL Calls being made to Populate the Object
echo 'Done'; //execution continues
```

但是 Magento 不是这样的，它采用的是 Lazy Loading。延迟加载意味着在程序需要数据前，sql 是不执行的，如下。

```
$collection_of_products = Mage::getModel('catalog/product')
```

```
->getCollection();
```

在这个时候 Magento 还没有链接数据库，你可以放心地去做你想要做的事。

```
$collection_of_products = Mage::getModel('catalog/product')
->getCollection();
$collection_of_products->addAttributeToSelect('meta_title');
```

你不必担心每次添加属性的时候 Magento 都会执行一个 sql，去获取数据，sql 只有在你需要数据的时候才会被执行。
Magento 对数据库连接层做了良好的封装，当然它也考虑到了效率问题。在一般情况下，你没必要去担心 sql 后台是怎么执行的，只需要专心做你的功能，例如区块、布局等。这是 Magento 非常优秀的地方。

过滤数据（Filtering Database Collections）

最重要的一个方法是 addFieldToFilter。通过这个方法可以添加我们 sql 中的 WHERE 语句。

```
public function testAction()
{
    $collection_of_products = Mage::getModel('catalog/product')
    ->getCollection();
    $collection_of_products->addFieldToFilter('sku','n2610');

    //another neat thing about collections is you can pass them into the
count     //function.  More PHP5 powered goodness
    echo "Our collection now has " . count($collection_of_products) . '
item(s)';
    var_dump($collection_of_products->getFirstItem()->getData());
}
```

addFieldToFilter 方法中的第一个参数是你想过滤的字段名称，第二个是你想过滤的值。例如刚刚 sku 是字段名称，n2610 是值。
第二个参数也可以被用来指定某一类型的数据。稍微有些复杂，我们继续往下看。

```
$collection_of_products->addFieldToFilter('sku','n2610');
```

这个等同于 sql 中的 where 条件句

```
WHERE sku = "n2610"
```

下面的例子自己尝试下

```
public function testAction()
{
    var_dump(
    (string)
    Mage::getModel('catalog/product')
    ->getCollection()
    ->addFieldToFilter('sku','n2610')
    ->getSelect());
```

```
}
```

将会输出这个

```
SELECT `e`.* FROM `catalog_product_entity` AS `e` WHERE (e.sku =
'n2610')'
```

但是这个很快会变得很复杂。试着做下面的练习。

```
var_dump(
(string)
Mage::getModel('catalog/product')
->getCollection()
->addAttributeToSelect('*')
->addFieldToFilter('meta_title','my title')
->getSelect()
);
```

输出的将是下面的 sql 语句。

```
SELECT `e`.*, IF(_table_meta_title.value_id>0,
_table_meta_title.value, _table_meta_title_default.value) AS
`meta_title`
FROM `catalog_product_entity` AS `e`
INNER JOIN `catalog_product_entity_varchar` AS
`_table_meta_title_default`
   ON (_table_meta_title_default.entity_id = e.entity_id) AND
(_table_meta_title_default.attribute_id='103')
   AND _table_meta_title_default.store_id=0
LEFT JOIN `catalog_product_entity_varchar` AS `_table_meta_title`
   ON (_table_meta_title.entity_id = e.entity_id) AND
(_table_meta_title.attribute_id='103')
   AND (_table_meta_title.store_id='1')
WHERE (IF(_table_meta_title.value_id>0, _table_meta_title.value,
_table_meta_title_default.value) = 'my title')
```

在你有空的时候可以好好研究下上面的 sql 语句，我们先不转移焦点，继续我们下面的讲解。

其它比较运算符

我确定在刚刚的练习中，你想知道如何实现一个不是“=”的 where 条件句，例如不等于、大于、小于。刚刚我们有讲过 addFieldToFilter 的第二个参数允许传入不同“类型”。

其实很简单，只要将一个简单的数组作为第二个参数传入 addFieldToFilter 方法就可以变换条件句。

数组的键就是“类型”，关联的值就是你想过滤的值。我们改写下上面的代码。

```
public function testAction()
{
    var_dump(
    (string)
    Mage::getModel('catalog/product')
    ->getCollection()
```

```
    ->addFieldToFilter('sku',array('eq'=>'n2610'))
    ->getSelect()
    );
}
```

看上面的过滤器

```
addFieldToFilter('sku',array('eq'=>'n2610'))
```

正如你看到的，第二个参数是一个 php 的数组。它的键是“eq”，代表等于的意思。
Magento 在这个函数中有一系列英语的缩写，这些词的资料可以参考《tear of remembrance》。
这些沿用了 Perl 语言中的一些比较运算符号。
在这里我将 Magento 所有的条件判断符号列出来供大家参考。

```
array("eq"=>'n2610')
WHERE (e.sku = 'n2610')

array("neq"=>'n2610')
WHERE (e.sku != 'n2610')

array("like"=>'n2610')
WHERE (e.sku like 'n2610')

array("nlike"=>'n2610')
WHERE (e.sku not like 'n2610')

array("is"=>'n2610')
WHERE (e.sku is 'n2610')

array("in"=>array('n2610'))
WHERE (e.sku in ('n2610'))

array("nin"=>array('n2610'))
WHERE (e.sku not in ('n2610'))

array("notnull"=>'n2610')
WHERE (e.sku is NOT NULL)

array("null"=>'n2610')
WHERE (e.sku is NULL)

array("gt"=>'n2610')
WHERE (e.sku > 'n2610')

array("lt"=>'n2610')
WHERE (e.sku < 'n2610')

array("gteq"=>'n2610')
```

```
WHERE (e.sku >= 'n2610')

array("moreq"=>'n2610') //a weird, second way to do greater than equal
WHERE (e.sku >= 'n2610')

array("lteq"=>'n2610')
WHERE (e.sku <= 'n2610')

array("finset"=>array('n2610'))
WHERE (find_in_set('n2610',e.sku))

array('from'=>'10','to'=>'20')
WHERE e.sku >= '10' and e.sku <= '20'
```

其中大多数是自我的理解，但有几个得特别注意。

in, nin, find_in_set

in and nin 条件句中，语序你传入一个数组作为值。例如：

```
array("in"=>array('n2610','ABC123')
WHERE (e.sku in ('n2610','ABC123'))
```

notnull, null

关键字 NULL 是最特殊的 sql 句，它将忽略你传入的值。

```
array("notnull"=>'n2610')
WHERE (e.sku is NOT NULL)
```

from – to 过滤

这是另一种过滤方式，在传入的数组中，允许你传入两个键，是从哪里到哪里的意思，一个数值区间。

```
public function testAction
{
        var_dump(
        (string)
        Mage::getModel('catalog/product')
        ->getCollection()
        ->addFieldToFilter('price',array('from'=>'10','to'=>'20'))
        ->getSelect()
        );
}
```

上面等同于

```
WHERE (_table_price.value >= '10' AND _table_price.value <= '20')
```

AND 或者 OR

根据刚才讲的内容，你可以知道，通过多个 addFieldToFilter 方法可以获得一个”AND”的条件句。

```
function testAction()
{
        echo(
        (string)
        Mage::getModel('catalog/product')
        ->getCollection()
        ->addFieldToFilter('sku',array('like'=>'a%'))
        ->addFieldToFilter('sku',array('like'=>'b%'))
        ->getSelect()
        );
}
```

等同于下面的子句

```
WHERE (e.sku LIKE 'a%') AND (e.sku LIKE 'b%')
```

但是，聪明的你可以发现，上面的例子不可能返回任何结果，因为一个 sku 不可能以 a 开头，同时也以 b 开头。
我们希望用的应该是”OR”，那么如何实现呢？这又使我们将焦点集中到了 addFieldToFilter 方法的第二个参数上。
如果你希望构造一个 or 的语句，首先我们构造两个参数。

```
public function testAction()
{
        $filter_a = array('like'=>'a%');
        $filter_b = array('like'=>'b%');
}
```

然后将它们作为一组参数传入 addFieldToFilter 方法中，如下。

```
public function testAction()
{
        $filter_a = array('like'=>'a%');
        $filter_b = array('like'=>'b%');
        echo(
        (string)
        Mage::getModel('catalog/product')
        ->getCollection()
        ->addFieldToFilter('sku',array($filter_a,$filter_b))
        ->getSelect()
        );
}
```

你可以看到这样的一个子句。

```
WHERE (((e.sku LIKE 'a%') OR (e.sku LIKE 'b%')))
```

总结

恭喜你，你现在已经是一个很不错的 Magento 开发者了！因为你不需要写任何 sql 语句，就可以获取几乎所有模型的所有你想要的数据。